The Power of Priority Channel Systems *

Christoph Haase Sylvain Schmitz Philippe Schnoebelen

LSV, ENS Cachan & CNRS, France

Abstract

We introduce Priority Channel Systems, a new class of channel systems where messages carry a numeric priority and where higher-priority messages can supersede lower-priority messages preceding them in the fifo communication buffers. The decidability of safety and inevitability properties is shown via the introduction of a *priority embedding*, a well-quasi-ordering that has not previously been used in well-structured systems. We then show how Priority Channel Systems can compute Fast-Growing functions and prove that the aforementioned verification problems are $\mathbf{F}_{\boldsymbol{\varepsilon_0}}$-complete.

1 Introduction

Channel systems are a family of distributed models where concurrent agents communicate via (usually unbounded) fifo communication buffers, called "channels". These models are well-suited for the formal specification and algorithmic analysis of communication protocols and concurrent programs (Boigelot and Godefroid, 1999; Bouajjani and Habermehl, 1999; Cécé and Finkel, 2005). They are also a fundamental model of computation, closely related to Post's tag systems.

A particularly interesting class of channel systems are the so-called *lossy channel systems (LCSs)*, where channels are unreliable and may lose messages (Cécé et al., 1996; Abdulla and Jonsson, 1996; Bouyer et al., 2012). For LCSs, several important behavioral properties, like safety or inevitability, are decidable. This is because these systems are *well-structured*: transitions are monotonic wrt. a (decidable) well-quasi-ordering of the configuration space (Abdulla et al., 2000; Finkel and Schnoebelen, 2001). Beyond their applications in verification, LCSs have turned out to be an important automata-theoretic tool for decidability or hardness in areas like Timed Automata, Metric Temporal Logic, modal logics, etc. (Abdulla et al., 2005; Kurucz, 2006; Ouaknine and Worrell, 2007; Lasota and Walukiewicz, 2008). They are also a fundamental model of computation capturing the $\mathbf{F}_{\omega^\omega}$-complexity level in Wainer *et al.*'s Fast-Growing Hierarchy, see (Chambart and Schnoebelen, 2008; Schmitz and Schnoebelen, 2011, 2012).

Despite their wide applicability, LCSs reveal shortcomings when applied to modeling systems or protocols that treat messages discriminatingly according

*Work partially funded by the ReacHard project ANR 11 BS02 001 01.

to some specified rule set. An example is the prioritisation of messages, which is central to ensuring *quality of service* (QoS) properties in networking architectures, and is usually implemented by allowing for tagging messages with some relative priority. For instance, the Differentiated Services (DiffServ) architecture, described in RFC 2475, allows for a field specifying the relative priority of an IP packet with respect to a finite set of priorities, and network links may decide to arbitrarily drop IP packets of lower priority in favor of higher priority packets once the network congestion reaches a critical point.

Our contributions In this paper, we introduce *Priority Channel Systems*, or PCSs for short, a family of channel systems where each message is equipped with a priority level, and where higher-priority messages can supersede lower-priority messages (that are dropped). Our model abstracts from the contents of messages by just considering the priority levels (but see App. D for a generalization to infinite alphabets of message contents). We show that PCSs are well-structured when configurations are ordered by the *(prioritized) superseding ordering*, a new well-quasi-ordering that is closely related to the gap-embedding of (Schütte and Simpson, 1985). This entails the decidability of safety and termination (among other properties) for PCSs.

Using techniques from (Schmitz and Schnoebelen, 2011; Schütte and Simpson, 1985), the proof that the superseding ordering is a well-quasi-ordering gives an $\mathbf{F}_{\varepsilon_0}$ upper bound on the complexity of PCS verification, far higher than the $\mathbf{F}_{\omega^\omega}$-complete complexity of LCSs.

In the second part of this paper, we prove a matching lower bound: building upon ideas and techniques developed for less powerful models (Chambart and Schnoebelen, 2008; Schnoebelen, 2010a; Haddad et al., 2012), we show how PCSs can robustly simulate the computation of Fast Growing Functions F_α (and their inverses) for all ordinals α up to ε_0.

Along the way we show how some other well-quasi-ordered data structures, e.g. trees with strong embedding, can be reflected in strings with priority ordering, opening the way to $\mathbf{F}_{\varepsilon_0}$ upper bounds in other areas of algorithmic verification.

2 Priority Channel Systems

We define Priority Channel Systems as consisting of a single process since this is sufficient for our purposes in this paper.[1]

For every $d \in \mathbb{N}$, the *level-d priority alphabet* is $\Sigma_d \stackrel{\text{def}}{=} \{0, 1, \ldots, d\}$. A *level-d priority channel system* (a "d-PCS") is a tuple $S = (\Sigma_d, \mathtt{Ch}, Q, \Delta)$ where Σ_d is as above, $\mathtt{Ch} = \{\mathtt{c}_1, \ldots, \mathtt{c}_m\}$ is a set of m *channel names*, $Q = \{q_1, q_2, \ldots\}$ is a finite set of *control states*, and $\Delta \subseteq Q \times \mathtt{Ch} \times \{!, ?\} \times \Sigma_d \times Q$ is a set of *transition rules* (see below).

[1] Obviously, systems that are more naturally seen as made up of several concurrent components can be represented by a single process obtained as an asynchronous product of the components.

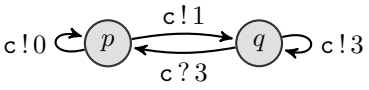

Figure 1: A simple single-channel 3-PCS.

2.1 Semantics

The operational semantics of a PCS S is given under the form of a transition system. We let $Conf_S \stackrel{\text{def}}{=} Q \times (\Sigma_d^*)^m$ be the set of all configurations of S, denoted $C, D, \ldots$ A configuration $C = (q, x_1, \ldots, x_m)$ records an instantaneous control point (a state in Q) and the contents of the m channels (sequences of messages from Σ_d). A sequence $x \in \Sigma_d^*$ has the form $x = a_1 \ldots a_\ell$ and we let $\ell = |x|$. Concatenation is denoted multiplicatively, with ε denoting the empty sequence.

The labeled transition relation between configurations, denoted $C \xrightarrow{\delta} C'$, is generated by the rules in $\Delta = \{\delta_1, \ldots, \delta_k\}$. It is actually convenient to define three such transition relations, denoted $\rightarrow_{\text{rel}}$, $\rightarrow_{\text{w}}$, and $\rightarrow_{\#}$ respectively.

Reliable Semantics We start with $\rightarrow_{\text{rel}}$ that corresponds to "reliable" steps, or more correctly steps with no superseding of lower-priority messages. As is standard, for a *reading rule* of the form $\delta = (q, \mathsf{c}_i, ?, a, q') \in \Delta$, there is a step $C \xrightarrow{\delta}_{\text{rel}} C'$ if $C = (q, x_1, \ldots, x_m)$ and $C' = (q', y_1, \ldots, y_m)$ for some $x_1, y_1, \ldots, x_m, y_m$ such that $x_i = a y_i$ and $x_j = y_j$ for all $j \neq i$, while for a *writing rule* $\delta = (q, \mathsf{c}_i, !, a, q') \in \Delta$, there is a step $C \xrightarrow{\delta}_{\text{rel}} C'$ if $y_i = x_i a$ (and $x_j = y_j$ for all $j \neq i$). These "reliable" steps correspond to the behavior of queue automata, or (reliable) channel systems, a Turing-powerful computation model.

Write-Superseding The actual behavior of PCSs, denoted $\rightarrow_{\text{w}}$, is best defined as a modification of $\rightarrow_{\text{rel}}$, and more precisely by modifying the semantics of writing rules. Formally, for $\delta = (q, \mathsf{c}_i, !, a, q') \in \Delta$, and for C, C' as above, there is a step $C \xrightarrow{\delta}_{\text{w}} C'$ if $y_i = za$ for a factorization $x_i = zz'$ of x_i where $z' \in \Sigma_a^*$, i.e., where z' only contains messages from the level-a priority subalphabet. In other words, after $\mathsf{c}_i!a$, the channel will contain a sequence y_i obtained from x_i by appending a in a way that may drop (erase) any number of suffix messages with priority $\leq a$, hence the "$z' \in \Sigma_a^*$" requirement. (And $x_j = y_j$ for all $j \neq i$.) Reading steps are unchanged so that $C \xrightarrow{\delta}_{\text{rel}} C'$ implies $C \xrightarrow{\delta}_{\text{w}} C'$. This gives rise to a transition system $\mathcal{S}_{\text{w}} \stackrel{\text{def}}{=} (Conf_S, \rightarrow_{\text{w}})$.

For example, the PCS from Figure 1 has the following run:

$$p, 02\underline{00} \xrightarrow{!1}_{\text{w}} q, 0\underline{21} \xrightarrow{!3}_{\text{w}} q, 03 \xrightarrow{!3}_{\text{w}} q, \underline{033} \xrightarrow{!3}_{\text{w}} q, 3 \xrightarrow{?3}_{\text{w}} p, \varepsilon$$

where in every configuration we underline the messages that will be superseded in the next step (and where, for simplicity, we do not write the full rule δ on the steps). Note that, as specified in the semantics, the first step could not be "$(p, \underline{0200}) \xrightarrow{!1}_{\text{w}} (q, 21)$": the written 1 is not allowed to supersede the higher-priority 2 *hence it cannot supersede the* 0 *that is earlier in the channel.*

Internal-Superseding There is another semantics for priorities, obtained by extending reliable steps with *internal superseding steps*, denoted $C \xrightarrow{\mathsf{c}_i \# k}_{\#} C'$, which can be performed at any time in an uncontrolled manner.

Formally, for two words $x, y \in \Sigma_d^*$ and $k \in \mathbb{N}$, we write $x \xrightarrow{\#k}_{\#} y \overset{\text{def}}{\Leftrightarrow} x$ is some $a_1 \dots a_\ell$, $1 \leq k < |x| = \ell$, $a_k \leq a_{k+1}$ and $y = a_1 \dots a_{k-1} a_{k+1} \dots a_\ell$. In other words, the k-th message in x is superseded by its immediate successor a_{k+1}, with the condition that a_k is not of higher priority. We write $x \to_{\#} y$ when $x \xrightarrow{\#k}_{\#} y$ for some k, and use $x \leftarrow_{\#} y$ when $y \to_{\#} x$. The transitive reflexive closure $\overset{*}{\leftarrow}_{\#}$ is called the *superseding* ordering and is denoted by $\leq_{\#}$. Put differently, $\to_{\#}$ is a rewrite relation over Σ_d^* according to the rules $\{aa' \to a' \mid 0 \leq a \leq a' \leq d\}$.

This is extended to steps between configurations by $C = (q, x_1, \dots, x_m) \xrightarrow{\mathsf{c}_i \# k}_{\#} C' = (q', y_1, \dots, y_m) \overset{\text{def}}{\Leftrightarrow} q = q'$ and $x_i \xrightarrow{\#k}_{\#} y_i$ (and $x_j = y_j$ for $j \neq i$). Furthermore, every reliable step is a valid step: for any rule δ, $C \xrightarrow{\delta}_{\#} C'$ iff $C \xrightarrow{\delta}_{\text{rel}} C'$, giving rise to a second transition system associated with S: $\mathcal{S}_{\#} \overset{\text{def}}{=} (Conf_S, \to_{\#})$. E.g., the PCS from Fig. 1 can perform

$$p, 0200 \xrightarrow{!1}_{\#} q, 02\underline{0}01 \xrightarrow{\#3}_{\#} q, \underline{0}201 \xrightarrow{\#1}_{\#} q, 2\underline{0}1 \xrightarrow{\#2}_{\#} q, 21$$

while, as we noted earlier, $(p, 0200) \overset{*}{\not\to}_{\text{w}} (q, 21)$.

2.2 Relating the Superseding Semantics

The Write-Superseding semantics adopts a localized viewpoint, where a single system or protocol manages several priority levels for its communication through a fifo channel that can be congested.

The Internal-Superseding semantics allows superseding to occur at any time (not just when writing) and anywhere in the channel. It is appropriate when abstracting from situations where end-to-end communication actually goes through a series of consecutive relays, network switches and buffers, each of them possibly handling the incoming traffic with a Write-Superseding policy.

When developing the formal theory of PCSs, $\mathcal{S}_{\#}$, the Internal-Superseding semantics, is more liberal and harder to control than $\mathcal{S}_{\text{w}}$. It is also finer-grained than $\mathcal{S}_{\text{w}}$ (superseding occurs one message at a time) but this is less significant.

The consequence is that, in practice, it is usually easier to design a correct PCS (and proving its correctness) when one assumes the Write-Superseding semantics—as we do in Section 6—, while it is easier to develop the formal theory of PCSs with the Internal-Superseding semantics—as we do next. However, the two semantics are, in a sense, equivalent since $\mathcal{S}_{\#}$ and $\mathcal{S}_{\text{w}}$ simulate one another:

Proposition 1 (See App. A)**.** *Let $C_0 = (q, \varepsilon, \dots, \varepsilon)$ be a configuration with empty channels, and C_f be any configuration. Then $C_0 \xrightarrow{+}_w C_f$ if, and only if, $C_0 \xrightarrow{+}_{\#} C_f$.*

We conclude this discussion by observing that PCSs can simulate lossy channel systems (in fact they can simulate the dynamic lossy channel systems and the timed lossy channel systems of (Abdulla et al., 2012), see App. B). Hence reachability and termination (see Thm. 2) are at least $\mathbf{F}_{\omega^\omega}$-hard for PCSs, and problems like boundedness or repeated control-state reachability (see (Schnoebelen, 2010b) for more) are undecidable for them.

Remark 1 (A stricter policy?). It is possible to define a stricter policy for priorities where a higher-priority message may only supersede messages with *strictly* lower priority. Write $x \xrightarrow{\#k}_{\succ} y$ when $x \xrightarrow{\#k}_{\#} y$ and $x = a_1 \dots a_\ell$ has $a_k < a_{k+1}$. This semantics is natural in some situations but the resulting model is Turing-powerful (see App. B) and not amenable to the wqo-based algorithmic techniques we develop for PCSs.

2.3 Priority Channel Systems are Well-Structured

Our main result regarding the verification of PCSs is that they are *well-structured* systems. Recall that $C \leq_{\#} D \overset{\text{def}}{\Leftrightarrow} C$ is some $(p, y_1, \dots, y_m)$ and D is $(p, x_1, \dots, x_m)$ with $x_i \leq_{\#} y_i$ for $i = 1, \dots, m$, or equivalently, C can be obtained from D by internal superseding steps.

Theorem 1 (PCSs are WSTSs)**.** *For any PCS S, the transition system $\mathcal{S}_{\#}$ with configurations ordered by $\leq_{\#}$ is a well-structured transition system (with stuttering compatibility).*

Proof. There are two conditions to check:

1. **wqo:** $(Conf_S, \leq_{\#})$ is a well-quasi-ordering as will be shown next (see Thm. 3 in Section 3).

2. **monotonicity:** Checking stuttering compatibility (see (Finkel and Schnoebelen, 2001, def. 4.4)) is trivial with the $\leq_{\#}$ ordering. Indeed, assume that $C \leq_{\#} D$ and that $C \rightarrow_{\#} C'$ is a step from the "smaller" configuration. Then in particular $D \xrightarrow{*}_{\#} C$ by definition of $\rightarrow_{\#}$, so that clearly $D \xrightarrow{+}_{\#} C'$ and D can simulate any step from C. □

Observe that it would not be so easy to prove well-structuredness for $\mathcal{S}_{\text{w}}$ (to begin with, another ordering would be required).

A consequence of the well-structuredness of PCSs is the decidability of several natural verification problems. In this paper we focus on "Reachability"[2] (given a PCS, an initial configuration C_0, and a set of configurations $G \subseteq Conf_S$, does $C_0 \xrightarrow{*}_{\#} D$ for some $D \in G$?), and "Inevitability" (do all maximal runs from C_0 eventually visit G?) which includes "Termination" as a special case.

Theorem 2 (Verifying PCSs)**.** *Reachability and Inevitability are decidable for PCSs with Internal-Superseding semantics.*

Proof (Sketch). The generic WSTS algorithms (Finkel and Schnoebelen, 2001) apply after we check the minimal effectivity requirements: the ordering $\leq_{\#}$ between configurations is decidable (even in NLogSpace, see Section 3.2) and the operational semantics is finitely branching and effective (one can compute the immediate successors of a configuration, and the minimal immediate predecessors of an upward-closed set).

We note that Reachability and Coverability coincide (even for zero-length runs when C_0 has empty channels) since $\xrightarrow{+}_{\#}$ coincides with $\geq_{\#} \circ \xrightarrow{+}_{\#}$, and that the answer to a Reachability question only depends on the (finitely many)

[2] Also called "Safety" when we want to check that G is *not* reachable.

minimal elements of G. One can even compute $Pre^*(G)$ for G given, e.g., as a regular subset of $Conf_S$.

For Inevitability, the algorithms in (Abdulla et al., 2000; Finkel and Schnoebelen, 2001) assume that G is downward-closed but, in our case where $\overset{+}{\to}_\#$ and $\geq_\# \circ \overset{+}{\to}_\#$ coincide, decidability can be shown for arbitrary (recursive) G, as in (Schnoebelen, 2010b, Thm. 4.4). □

Remark 2. With Prop. 1 and standard coding tricks, Thm. 2 directly provides decidability for Reachability and Termination when one assumes Write-Superseding semantics.

3 Priority Embedding

This section focuses on the superseding ordering $\leq_\#$ on words and establishes the fundamental properties we use for reasoning about PCSs. Recall that $\leq_\# \overset{\text{def}}{=} \overset{*}{\leftarrow}_\#$, the reflexive transitive closure of the inverse of $\to_\#$; we prove that $(\Sigma_p^*, \leq_\#)$ is a *well-quasi-ordering* (a wqo). Recall that a quasi-ordering $(X, \preccurlyeq)$ is a wqo if any infinite sequence $x_0, x_1, x_2, \ldots$ over X contains an infinite increasing subsequence $x_{i_0} \preccurlyeq x_{i_1} \preccurlyeq x_{i_2} \preccurlyeq \cdots$

3.1 Embedding with Priorities

For two words $x, y \in \Sigma_d^*$, we let $x \sqsubseteq_\text{p} y \overset{\text{def}}{\Leftrightarrow} x = a_1 \cdots a_\ell$ and y can be factored as $y = z_1 a_1 z_2 a_2 \cdots z_\ell a_\ell$ with $z_i \in \Sigma_{a_i}^*$ for $i = 1, \ldots, \ell$. For example, $201 \sqsubseteq_\text{p} 22011$ but $120 \not\sqsubseteq_\text{p} 10210$ (factoring 10210 as $z_1 1 z_2 2 z_3 0$ needs $z_3 = 1 \notin \Sigma_0^*$). If $x \sqsubseteq_\text{p} y$ then x is a subword of y and x can be obtained from y by removing factors of messages with priority not above the first preserved message to the right of the factor. In particular, $x \sqsubseteq_\text{p} y$ implies $y \overset{*}{\to}_\# x$, i.e., $x \leq_\# y$.

The definition immediately yields:

$$\varepsilon \sqsubseteq_\text{p} y \text{ iff } y = \varepsilon\,, \tag{1}$$

$$x_1 \sqsubseteq_\text{p} y_1 \text{ and } x_2 \sqsubseteq_\text{p} y_2 \text{ imply } x_1 x_2 \sqsubseteq_\text{p} y_1 y_2\,, \tag{2}$$

$$x_1 x_2 \sqsubseteq_\text{p} y \text{ imply } \exists y_1 \sqsupseteq_\text{p} x_1 : \exists y_2 \sqsupseteq_\text{p} x_2 : y = y_1 y_2\,. \tag{3}$$

Lemma 1. *$(\Sigma_d^*, \sqsubseteq_p)$ is a quasi-ordering (i.e., is reflexive and transitive).*

Proof. Reflexivity is obvious from the definition. For transitivity, consider $x' \sqsubseteq_\text{p} x \sqsubseteq_\text{p} y$ with $x = a_1 \cdots a_\ell$ and $y = z_1 a_1 \cdots z_\ell a_\ell$. In view of Eqs. (1–3) it is enough to show $x' \sqsubseteq_\text{p} y$ in the case where $|x'| = 1$. Consider then $x' = a$. Now $x' \sqsubseteq_\text{p} x$ implies $a = a_\ell$ and $a \geq a_i$, hence $\Sigma_{a_i}^* \subseteq \Sigma_a^*$, for all $i = 1, \ldots, \ell$. Letting $z \overset{\text{def}}{=} z_1 a_1 \cdots z_{\ell-1} a_{\ell-1} z_\ell$ yields $y = za$ for $z \in \Sigma_a^*$. Hence $x' \sqsubseteq_\text{p} z$. □

We can now relate superseding and priority orderings with:

Proposition 2. *For all $x, y \in \Sigma_d^*$, $x \sqsubseteq_p y$ iff $x \leq_\# y$.*

Proof. Obviously, $y \overset{\#k}{\to}_\# x$ allows $x \sqsubseteq_\text{p} y$ with z_k being the superseded message (and $z_i = \varepsilon$ for $i \neq k$), so that $\leq_\#$ is included in $\sqsubseteq_\text{p}$ by Lem. 1. In the other direction $x \sqsubseteq_\text{p} y$ entails $x \leq_\# y$ as noted earlier. □

3.2 Canonical Factorizations and Well-quasi-ordering

For our next development, we define the *height*, written $h(x)$, of a sequence $x \in \Sigma_d^*$ as being the highest priority occurring in x (by convention, we let $h(\varepsilon) \stackrel{\text{def}}{=} -1$). Thus, $x \in \Sigma_h^*$ iff $h \geq h(x)$. (We further let $\Sigma_{-1} \stackrel{\text{def}}{=} \emptyset$.) Any $x \in \Sigma_d^*$ has a unique *canonical* factorization $x = x_0 h x_1 h \cdots x_{k-1} h x_k$ where k is the number of occurrences of $h = h(x)$ in x and where the $k+1$ *residuals* x_0, $x_1, \ldots, x_k$ are in Σ_{h-1}^*.

The point of this decomposition is the following sufficient condition for $x \sqsubseteq_{\text{p}} y$.

Lemma 2. *Let $x = x_0 h \cdots h x_k$ and $y = y_0 h \cdots h y_m$ be canonical factorizations with $h = h(x) = h(y)$. If there is a sequence $0 = j_0 < j_1 < j_2 < \cdots < j_{k-1} < j_k = m$ of indexes s.t. $x_i \sqsubseteq_p y_{j_i}$ for all $i = 0, \ldots, k$ then $x \sqsubseteq_p y$.*

Proof. We show $x \leq_\# y$. Note that $h y_i h \xrightarrow{*}_\# h$ for all $i = 1, \ldots, m$, so $y \xrightarrow{*}_\# y' \stackrel{\text{def}}{=} y_{j_0} h y_{j_1} h y_{j_2} \cdots h y_{j_k}$ (recall that $0 = j_0$ and $m = j_k$). From $x_i \sqsubseteq_{\text{p}} y_{j_i}$ we deduce $y_{j_i} \xrightarrow{*}_\# x_i$ for all $i = 0, \ldots, k$, hence $y' \xrightarrow{*}_\# x_0 h \cdots h x_k = x$. □

The condition in the statement of Lemma 2 is usually written $\langle x_0, \ldots, x_k \rangle \preceq_* \langle y_0, \ldots, y_m \rangle$, using the *sequence extension* of $\sqsubseteq_{\text{p}}$ on sequences of residuals.

Theorem 3. *$(\Sigma_d^*, \sqsubseteq_p)$ is a well-quasi-ordering (a wqo).*

Proof. By induction on d. The base case $d = -1$ is trivial since Σ_{-1}^* is $\emptyset^* = \{\varepsilon\}$, a singleton. For the induction step, consider an infinite sequence $x_0, x_1, \ldots$ over Σ_d^*. We can extract an infinite subsequence, where all x_i's have the same height h (since $h(x_i)$ is in a finite set) and, since the residuals are in Σ_{d-1}^*, a wqo by ind. hyp., further extract an infinite subsequence where the first and the last residuals are increasing, i.e., $x_{i_0,0} \sqsubseteq_{\text{p}} x_{i_1,0} \sqsubseteq_{\text{p}} x_{i_2,0} \sqsubseteq_{\text{p}} \cdots$ and $x_{i_0,k_0} \sqsubseteq_{\text{p}} x_{i_1,k_1} \sqsubseteq_{\text{p}} x_{i_2,k_2} \sqsubseteq_{\text{p}} \cdots$. Now recall that, by Higman's Lemma, the sequence extension $((\Sigma_{d-1}^*)^*, \preceq_*)$ is a wqo since, by ind. hyp., $(\Sigma_{d-1}^*, \sqsubseteq_{\text{p}})$ is a wqo. We may thus further extract an infinite subsequence that is increasing for $\preceq_*$ on the residuals, i.e., with $\langle x_{i_0,0}, x_{i_0,1}, \ldots, x_{i_0,k_0} \rangle \preceq_* \langle x_{i_1,0}, x_{i_1,1}, \ldots, x_{i_1,k_1} \rangle \preceq_* \langle x_{i_2,0}, x_{i_2,1}, \ldots, x_{i_2,k_2} \rangle \preceq_* \cdots$ With Lemma 2 we deduce $x_{i_0} \sqsubseteq_{\text{p}} x_{i_1} \sqsubseteq_{\text{p}} x_{i_2} \sqsubseteq_{\text{p}} \cdots$. Hence $(\Sigma_d^*, \sqsubseteq_{\text{p}})$ is a wqo. □

Remark 3. Thm. 3 and Prop. 2 prove that $\leq_\#$ is a wqo on configurations of PCSs, as we assumed in Section 2.3. There we also assumed that $\leq_\#$ is decidable. We can now see that it is in NLogSpace, since, in view of Prop. 2, one can check whether $x \leq_\# y$ by reading x and y simultaneously while guessing nondeterministically a factorization $z_1 a_1 \cdots z_\ell a_\ell$ of y, and checking that $z_i \in \Sigma_{a_i}^*$.

4 Applications of Priority Embedding to Trees

In this section we show how tree orderings can be reflected into sequences over a priority alphabet. This serves two purposes. First, it illustrates the "power" of priority embeddings, reproving that strong tree embeddings form a wqo as a byproduct. Second, the reflection defined will subsequently be used in Section 6 to provide an encoding of ordinals that PCSs can manipulate "robustly."

4.1 Encoding Bounded Depth Trees

Given an alphabet Γ, the set of finite, ordered, unranked labeled trees (aka variadic terms) over Γ, noted $T(\Gamma)$, is the smallest set such that, if f is in Γ and $t_1, \dots, t_n$ are $n \geq 0$ trees in $T(\Gamma)$, then the tree $f(t_1 \cdots t_n)$ is in $T(\Gamma)$. A *context* C is defined as usual as a tree with a single occurrence of a leaf labeled by a distinguished variable x. Given a context C and a tree t, we can form a tree $C[t]$ by plugging t instead of that x-labeled leaf.

Let d be a depth in $\mathbb{N}$ and $\bullet$ be a node label. We consider the set $T_d = T_d(\{\bullet\})$ of trees of depth at most d with $\bullet$ as single possible label; for instance, $T_0 = \{\bullet()\}$ contains a single tree, and the two trees shown in Figure 2 are in T_2:

Figure 2: Two trees in T_2.

It is a folklore result that one can encode bounded depth trees into finite sequences using canonical factorizations. Here we present a natural variant that is rather well-suited for our constructions in Section 6. We encode trees of bounded depth using the function $s_d : T_{d+1} \to \Sigma_d^*$ defined by induction on d as

$$s_d(\bullet(t_1 \cdots t_n)) \stackrel{\text{def}}{=} \begin{cases} \varepsilon & \text{if } n = 0, \\ s_{d-1}(t_1)d \cdots s_{d-1}(t_n)d & \text{otherwise.} \end{cases} \tag{4}$$

For instance, if we fix $d = 1$, the left tree in Figure 2 is encoded as "111" and the right one as "0011". Note that the encoding depends on the choice of d: for $d = 2$ we would have encoded the trees in Figure 2 as "222" and "1122", respectively.

Not every string in Σ_d^* is the encoding of a tree according to s_d: for $-1 \leq a \leq d$, we let $P_a \stackrel{\text{def}}{=} (P_{a-1}\{a\})^*$ be the set of *proper encodings of height* a, with further $P_{-1} \stackrel{\text{def}}{=} \{\varepsilon\}$. Then $P \stackrel{\text{def}}{=} \bigcup_{a \leq d} P_a$ is the set of *proper* words in Σ_d^*. A proper word x is either empty or belongs to a unique P_a with $a = h(x)$, and has then a canonical factorization of the form $x = x_1 a \cdots x_m a$ with every x_j in P_{a-1}. Put differently, a non-empty $x = a_1 \cdots a_\ell$ is in P_a if and only if $a_\ell = h(x)$ and $a_{i+1} - a_i \leq 1$ for all $i < \ell$ (we say that x *has no jumps*: along proper words, priorities only increase smoothly, but can decrease sharply). For example, 02 is not proper (it has a jump) while 012 is proper; 233123401234 is proper too.

Given a depth a, we see that s_a is a bijection between T_{a+1} and P_a, with the inverse defined by

$$\tau(\varepsilon) \stackrel{\text{def}}{=} \bullet() \,, \tag{5}$$

$$\tau(x = x_1 h(x) \cdots x_m h(x)) \stackrel{\text{def}}{=} \bullet(\tau(x_1) \cdots \tau(x_m)) \,. \tag{6}$$

4.2 Strong Tree Embeddings

One can provide a formal meaning to the notion of a wqo $(B, \preccurlyeq_B)$ being more powerful than another one $(A, \preccurlyeq_A)$ through *order reflections*, i.e. through the existence of a mapping $r : A \to B$ such that $r(x) \preccurlyeq_B r(y)$ implies $x \preccurlyeq_A y$ for all

x, y in A. Observe that if B reflects A, i.e., there is an order reflection from A to B, and $(B, \preccurlyeq_B)$ is a wqo, then $(A, \preccurlyeq_A)$ is necessarily a wqo. We show here that $(\Sigma_d^*, \sqsubseteq_{\mathrm{p}})$ reflects bounded-depth trees endowed with the strong tree-embedding relation.

Let t and t' be two trees in T_d. We say that t *strongly embeds* into t', written $t \sqsubseteq_T t'$, if it can be obtained from t' by deleting whole subtrees, i.e. $\sqsubseteq_T$ is the reflexive transitive closure of the relation $t \sqsubset_T^1 t' \stackrel{\text{def}}{\Leftrightarrow} t = C[\bullet(t_1 \cdots t_{i-1} t_{i+1} \cdots t_n)]$ and $t' = C[\bullet(t_1 \cdots t_{i-1} t_i t_{i+1} \cdots t_n)]$ for some context C and subtrees $t_1, \ldots, t_n$. Strong tree embeddings refine the *homeomorphic tree embeddings* used in Kruskal's Tree Theorem; in general they do not give rise to a wqo, but in the case of bounded depth trees they do. The two trees in Figure 2 are *not* related by any homeomorphic tree embedding, and thus neither by strong tree embedding.

Observe that the leaf $\bullet()$ strongly embeds into any other tree: $\bullet() \sqsubseteq_T t$ for all t. Let us consider the *extension* operation "@" on trees, which is defined for $n \geq 0$ by

$$\bullet(t_1 \cdots t_n) \mathbin{@} t \stackrel{\text{def}}{=} \bullet(t_1 \cdots t_n t) \,; \tag{7}$$

in particular, $\bullet() \mathbin{@} t = \bullet(t)$. Also observe that, if y is in P_a and z in P_{a-1}, then

$$\tau(yza) = \tau(y) \mathbin{@} \tau(z) \,. \tag{8}$$

Finally observe that $\sqsubseteq_T$ is a precongruence for @:

$$t_1 \sqsubseteq_T t_1' \text{ and } t_2 \sqsubseteq_T t_2' \text{ imply } t_1 \mathbin{@} t_2 \sqsubseteq_T t_1' \mathbin{@} t_2' \,, \tag{9}$$

$$t \sqsubseteq_T t \mathbin{@} t' \,. \tag{10}$$

Proposition 3. *The function s_d is an order reflection from $(T_{d+1}, \sqsubseteq_T)$ to $(\Sigma_d^*, \sqsubseteq_p)$.*

Proof. Let x and x' be two proper words in P_d with $x \sqsubseteq_{\mathrm{p}} x'$; we show by induction on x that $\tau(x) \sqsubseteq_T \tau(x')$. If x is empty, then $x \sqsubseteq_{\mathrm{p}} x'$ requires $x' = x$ Otherwise, we consider the canonical factorization $x = x_1 d \cdots x_k d z d$ for $k \geq 0$. Writing $y = x_1 d \cdots x_k d$, by (3), $x' = y'z'$ with $y \sqsubseteq_{\mathrm{p}} y'$ and $zd \sqsubseteq_{\mathrm{p}} z'$ where y' and z' are both in P_d. The canonical factorization of z' as $z_1' d \cdots z_m' d$ yields $z \sqsubseteq_{\mathrm{p}} z_1'$ with z_1' in P_{d-1}, as there is no other way of disposing of the other occurrences of d in z'. Then

$$\begin{aligned}
\tau(x) &= \tau(y) \mathbin{@} \tau(z) && \text{(by (8))} \\
&\sqsubseteq_T \tau(y') \mathbin{@} \tau(z_1') && \text{(by ind. hyp. and (9))} \\
&\sqsubseteq_T \tau(y') \mathbin{@} \tau(z_1') \mathbin{@} \cdots \mathbin{@} \tau(z_m') && \text{(by (10))} \\
&= \tau(x') \,. && \square
\end{aligned}$$

Corollary 1. *For each d, $(T_d, \sqsubseteq_T)$ is a wqo.*

4.3 Further Applications

As stated in the introduction to this section, our main interest in strong tree embeddings is in connection with structural orderings of ordinals; see Section 6. Bounded depth trees are also used in the verification of infinite-state systems as a means to obtain decidability results, in particular for tree pattern rewriting systems (Genest et al., 2008) in XML processing, and, using elimination trees (see Ossona de Mendez and Nešetřil, 2012), for bounded-depth graphs

used e.g. in the verification of ad-hoc networks (Delzanno et al., 2010), the π-calculus (Meyer, 2008), and programs (Bansal et al., 2013). These applications consider *labeled* trees, which are dealt with thanks to a generalization of $\sqsubseteq_{\mathrm{p}}$ to pairs (a, w) where a is a priority and w a symbol from some wqo $(\Gamma, \leq)$; see App. D.

This generalization of $\sqsubseteq_{\mathrm{p}}$ also allows to treat another wqo on trees, the *tree minor* ordering, using the techniques of Gupta (1992) to encode them in prioritized alphabets. The tree minor ordering is coarser than the homeomorphic embedding (e.g. in Figure 2, the left tree is a minor of the right tree), but the upside is that trees of unbounded depth can be encoded into strings.

The exact complexity of verification problems in the aforementioned models is currently unknown (Genest et al., 2008; Delzanno et al., 2010; Meyer, 2008; Bansal et al., 2013). Our encoding suggests them to be $\mathbf{F}_{\varepsilon_0}$-complete. We hope to see PCS Reachability employed as a "master" problem for $\mathbf{F}_{\varepsilon_0}$ for such results, like LCS Reachability for $\mathbf{F}_{\omega^\omega}$, which is used in reductions instead of more difficult proofs based on Turing machines and Hardy computations.

5 Fast-Growing Upper Bounds

The verification of infinite-state systems and WSTSs in particular turns out to require astronomic computational resources expressed as *subrecursive functions* (Löb and Wainer, 1970; Fairtlough and Wainer, 1998) of the input size. We show in this section how to bound the complexity of the algorithms presented in Section 2.3 and classify the Reachability and Inevitability problems using *fast-growing complexity classes* (Schmitz and Schnoebelen, 2012).

5.1 Subrecursive Hierarchies

Throughout this paper, we use *ordinal terms* inductively defined by the following grammar

$$(\Omega \ni) \quad \alpha, \beta, \gamma \ ::= \ 0 \mid \omega^\alpha \mid \alpha + \beta$$

where addition is associative, with 0 as the neutral element (the empty sum). Equivalently, we can then see a term other than 0 as a tree over the alphabet $\{+\}$; for instance the two trees in Figure 2 represent 3 and $\omega^2 + 1$ respectively, when putting the ordinal terms under the form $\alpha = \sum_{i=1}^{k} \omega^{\alpha_i}$. Such a term is 0 if $k = 0$, otherwise a *successor* if $\alpha_k = 0$ and a *limit* otherwise. We often write 1 as short-hand for ω^0, and ω for ω^1. The symbol λ is reserved for limit ordinal terms.

We can associate a set-theoretic ordinal $o(\alpha)$ to each term α by interpreting $+$ as the direct sum operator and ω as $\mathbb{N}$; this gives rise to a well-founded quasi-ordering $\alpha < \beta \overset{\text{def}}{\Leftrightarrow} o(\alpha) < o(\beta)$. A term $\alpha = \sum_{i=1}^{k} \omega^{\alpha_i}$ is in *Cantor normal form* (CNF) if $\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_k$ and each α_i is itself in CNF for $i = 1, \ldots, k$. Terms in CNF and set-theoretic ordinals below ε_0 are in bijection; it will however be convenient later in Section 6 to manipulate terms that are *not* in CNF.

With any limit term λ, we associate a *fundamental sequence* of terms $(\lambda_n)_{n \in \mathbb{N}}$, given by

$$\begin{aligned} (\gamma + \omega^{\beta+1})_n &\overset{\text{def}}{=} \gamma + \omega^\beta \cdot n = \gamma + \overbrace{\omega^\beta + \cdots + \omega^\beta}^{n}, \\ (\gamma + \omega^{\lambda'})_n &\overset{\text{def}}{=} \gamma + \omega^{\lambda'_n}. \end{aligned} \tag{11}$$

This yields $\lambda_0 < \lambda_1 < \cdots < \lambda_n < \cdots < \lambda$ for any λ, with furthermore $\lambda = \lim_{n\in\mathbb{N}} \lambda_n$. For instance, $\omega_n = n$, $(\omega^\omega)_n = \omega^n$, etc. Note that λ_n is in CNF when λ is.

We need to add a term ε_0 to Ω to represent the set-theoretic ε_0, i.e. the smallest solution of $x = \omega^x$. We take this term to be a limit term as well; we define the fundamental sequence for ε_0 by $(\varepsilon_0)_n \stackrel{\text{def}}{=} \Omega_n$, where for $n \in \mathbb{N}$, we use Ω_n as short-hand notation for the ordinal $\omega^{\omega^{\cdots^\omega}}\}n$ stacked ω's, i.e., for $\Omega_0 \stackrel{\text{def}}{=} 1$ and $\Omega_{n+1} \stackrel{\text{def}}{=} \omega^{\Omega_n}$.

Inner Recursion Hierarchies Our main subrecursive hierarchy is the *Hardy hierarchy*. Given a monotone expansive unary function $h\colon\mathbb{N}\to\mathbb{N}$, it is defined as an ordinal-indexed hierarchy of unary functions $(h^\alpha\colon\mathbb{N}\to\mathbb{N})_\alpha$ through

$$h^0(n) \stackrel{\text{def}}{=} n\,, \qquad h^{\alpha+1}(n) \stackrel{\text{def}}{=} h^\alpha\big(h(n)\big)\,, \qquad h^\lambda(n) \stackrel{\text{def}}{=} h^{\lambda_n}(n)\,.$$

Observe that h^1 is simply h, and more generally h^α is the αth iterate of h, using diagonalisation to treat limit ordinals.

A case of particular interest is to choose the successor function $H(n) \stackrel{\text{def}}{=} n+1$ for h. Then the *fast growing hierarchy* $(F_\alpha)_\alpha$ can be defined by $F_\alpha \stackrel{\text{def}}{=} H^{\omega^\alpha}$, resulting in $F_0(n) = H^1(n) = n+1$, $F_1(n) = H^\omega(n) = H^n(n) = 2n$, $F_2(n) = H^{\omega^2}(n) = 2^n n$ being exponential, $F_3 = H^{\omega^3}$ being non-elementary, $F_\omega = H^{\omega^\omega}$ being an Ackermannian function, F_{ω^k} a k-Ackermannian function, and $F_{\varepsilon_0} = H^{\varepsilon_0}\circ H$ a function whose totality is not provable in Peano arithmetic (Fairtlough and Wainer, 1998).

Fast-Growing Complexity Classes Our intention is to establish the "F_{ε_0} completeness" of verification problems on PCSs. In order to make this statement more precise, we define the class $\mathbf{F}_{\varepsilon_0}$ as a specific instance of the *fast-growing complexity classes* defined for $\alpha \geq 3$ by (see Schmitz and Schnoebelen, 2012, App. B)

$$\mathbf{F}_\alpha \stackrel{\text{def}}{=} \bigcup_{p\in\bigcup_{\beta<\alpha}\mathscr{F}_\beta} \textsc{DTime}(F_\alpha(p(n)))\,, \tag{12}$$

$$\mathscr{F}_\alpha = \bigcup_{c<\omega} \textsc{FDTime}(F_\alpha^c(n))\,, \tag{13}$$

where the class of functions $\mathscr{F}_\alpha$ as defined above is the αth level of the *extended Grzegorczyk hierarchy* (Löb and Wainer, 1970) when $\alpha \geq 2$; in particular, $\bigcup_{\alpha<\varepsilon_0}\mathscr{F}_\alpha$ is exactly the set of ordinal-recursive (aka "provably recursive") functions (Fairtlough and Wainer, 1998).

The complexity classes $\mathbf{F}_\alpha$ are naturally equipped with $\bigcup_{\beta<\alpha}\mathscr{F}_\beta$ as classes of reductions. For instance, $\mathscr{F}_2$ is the set of elementary functions, and $\mathbf{F}_3$ the class of problems with a tower of exponentials of height bounded by some elementary function of the input as an upper bound.[3]

[3]Note that, at such high complexities, the usual distinctions between deterministic vs. nondeterministic, or time-bounded vs. space-bounded computations become irrelevant.

5.2 Complexity Upper Bounds

Recall that an alternative characterization of a wqo $(X, \preccurlyeq)$ is that any sequence $x_0, x_1, x_2, \ldots$ over X verifying $x_i \not\preccurlyeq x_j$ for all $i < j$ is necessarily finite. Such sequences are called *bad*, and in order to bound the complexity of the algorithms from Thm. 2, we are going to bound the lengths of bad sequences over the wqo $(\mathit{Conf}_{\mathcal{S}}, \leq_{\#})$ using the *Length Function Theorem* of (Schmitz and Schnoebelen, 2011).

Let us explain the steps towards an upper bound for Termination in some detail; the results for Reachability and Inevitability are similar but more involved—see (Schmitz and Schnoebelen, 2012) for generic complexity arguments for WSTSs.

A Finite Witness Observe that, if an execution $C_0 \to_{\#} C_1 \to_{\#} C_2 \to_{\#} \cdots$ of the transition system $\mathcal{S}_{\#}$ verifies $C_i \leq_{\#} C_j$ for some indices $i < j$, then because $\mathcal{S}_{\#}$ is a WSTS, we can simulate the steps performed in this sequence after C_i but starting from C_j and build an infinite run. Conversely, if the system does not terminate, i.e. if there is an infinite execution $C_0 \to_{\#} C_1 \to_{\#} C_2 \to_{\#} \cdots$, then because of the wqo we will eventually find $i < j$ such that $C_i \leq_{\#} C_j$. Therefore, the system is non-terminating if and only if there is a finite witness of the form $C_0 \xrightarrow{*}_{\#} C_i \xrightarrow{+}_{\#} C_j$ with $C_i \leq_{\#} C_j$.

Controlled Sequences Another observation is that the size of successive configurations cannot grow arbitrarily along runs; in fact, the length of the channels contents can only grow by one symbol at a time using a write transition. This means that if we define $|C = (q, x_1, \ldots, x_m)| = \sum_{j=1}^{m} |x_j|$, then in an execution $C_0 \to_{\#} C_1 \to_{\#} C_2 \to_{\#} \cdots$, $|C_i| \leq |C_0| + i = H^i(|C_0|)$, i.e. any execution is *controlled* by the successor function H.

Maximal Order Type The last ingredient we need is a measure of the complexity of the wqo $(\mathit{Conf}_{\mathcal{S}}, \leq_{\#})$ called its *maximal order type*, which can be defined as the ordinal of its maximal linearization. We can bound o_d, the maximal order type of $(\Sigma_d^*, \sqsubseteq_{\mathrm{p}})$, by induction on d: $o_{-1} = 1$, and $o_d \leq \omega^{\omega^{o_{d-1}}} \cdot o_{d-1} \cdot o_{d-1} \cdot d$ using the order reflection implicit in the proof of Thm. 3 (see App. E for details). Therefore, the maximal order type $o_{\mathcal{S}}$ of $(\mathit{Conf}_{\mathcal{S}}, \leq_{\#})$ is bounded by $(\Omega_{2d+1})^m \cdot |Q|$.

Applying the Length Function Theorem Then, using the uniform upper bounds of (Schmitz and Schnoebelen, 2011), the maximal length of a bad execution in $\mathcal{S}_{\#}$ is bounded by $h^{o_{\mathcal{S}}}(|C_0|)$ for a fixed polynomial h. Setting $|S| = |\Delta| + |Q| + d + m$, we see that this length is less than $H^{\varepsilon_0}(p(|S| + |C_0|))$ for some fixed ordinal-recursive function p.

A Combinatory Algorithm Because the functions $(h^\alpha)_\alpha$ are space-constructible whenever h is, the above discussion yields a non-deterministic algorithm in $\mathbf{F}_{\varepsilon_0}$ for Termination: compute $L = h^{o_{\mathcal{S}}}(|C_0|)$ and look for an execution of length $L + 1$ in $\mathcal{S}_{\#}$. If one exists, it is necessarily a witness for nontermination; otherwise, the system is guaranteed to terminate from C_0.

We call this a *combinatory* algorithm, as it relies on the combinatory analysis provided by the Length Function Theorem to derive an upper bound on the size

of a finite witness for the property at hand—here Termination, but the same kind of techniques can be used for Reachability and Inevitability:

Theorem 4 (Complexity of PCS Verification)**.** *Reachability and Inevitability of PCSs are in* $\mathbf{F}_{\varepsilon_0}$.

6 Hardy Computations by PCSs

In this section we show how PCSs can weakly compute the Hardy functions H^α and their inverses for all ordinals α below Ω, which is the key ingredient for Thm. 5. For this, we develop (Section 6.1) encodings $s(\alpha) \in \Sigma_d^*$ for ordinals $\alpha \in \Omega_d$ and show how PCSs can compute with these codes, e.g. build the code for λ_n from the code of a limit λ. This is used (Section 6.2) to design PCSs that "*weakly compute*" H^α and $(H^\alpha)^{-1}$ in the sense of Def. 1 below.

6.1 Encoding Ordinals

Our encoding of ordinal terms as strings in Σ_d^* is exactly the encoding of trees presented in Section 4. For $0 \leq a \leq d$, we use the following equation to define the language $P_a \subseteq \Sigma_d^*$ of *proper encodings*, or just *codes*:

$$P_a \stackrel{\text{def}}{=} \varepsilon + P_a P_{a-1} a \,, \qquad\qquad P_{-1} \stackrel{\text{def}}{=} \varepsilon \,. \tag{14}$$

Let $P = P_{-1} + P_0 + \cdots + P_d$. Each P_a (and then P itself) is a regular language, with $P_a = (P_{a-1}a)^*$ as in Section 4; for instance, $P_0 = 0^*$.

Decompositions A code x is either the empty word ε, or belongs to a unique P_a. If $x \in P_a$ is not empty, it has a unique factorization $x = yza$ according to (14) with $y \in P_a$ and $z \in P_{a-1}$. The factor $z \in P_{a-1}$ in $x = yza$ can be developed further, as long as $z \neq \varepsilon$: a non-empty code $x \in P_d$ has a unique factorization as $x = y_d\, y_{d-1} \dots y_a\, a^\frown d$ with $y_i \in P_i$ for $i = a, \dots, d$, and where for $0 \leq a \leq b$, we write $a^\frown b$ for the *staircase* word $a(a+1)\cdots(b-1)b$, letting $a^\frown b = \varepsilon$ when $a > b$. We call this the *decomposition* of x. Note that the value of a is obtained by looking for the maximal suffix of x that is a staircase word. For example, $x = 23312340121234 \in P_4$ is a code and decomposes as

$$x = \overbrace{2331234}^{y_4}\overbrace{\varepsilon}^{y_3}\overbrace{012}^{y_2}\overbrace{\varepsilon}^{y_1}\overbrace{1234}^{1^\frown 4}.$$

Ordinal Encoding Following the tree encoding of Section 4, with a code $x \in P$, we associate an ordinal term $\eta(x)$ given by

$$\eta(\varepsilon) \stackrel{\text{def}}{=} 0 \,, \qquad\qquad \eta(yza) \stackrel{\text{def}}{=} \eta(y) + \omega^{\eta(z)} \,, \tag{15}$$

where $x = yza$ is the factorization according to (14) of $x \in P_a \setminus \{\varepsilon\}$. For example, $\eta(a) = \omega^0 = 1$ for all $a \in \Sigma_d$, $\eta(012) = \eta(234) = \omega^\omega$, and more generally $\eta(a^\frown b) = \Omega_{b-a}$. One sees that $\eta(x) < \Omega_{a+1}$ when $x \in P_a$.

The *decoding* function $\eta\colon P \to \Omega_{d+1}$ is onto (or surjective) but it is not bijective. However, it is a bijection between P_a and Ω_{a+1} for any $a \leq d$. Its

converse is the level-a *encoding* function $s_a\colon \Omega_{a+1} \to P_a$, defined with

$$s_a\Big(\sum_{i=1}^{p} \gamma_i\Big) \stackrel{\text{def}}{=} s_a(\gamma_1) \cdots s_a(\gamma_p)\,, \qquad\qquad s_a(\omega^\alpha) \stackrel{\text{def}}{=} s_{a-1}(\alpha)\, a\,.$$

Thus $s_a(0) = s(\sum \emptyset) = \varepsilon$ and, for example,

$$\begin{gathered} s_5(1) = 5\,, \qquad s_5(3) = 555\,, \qquad s_5(\omega) = 45\,, \\ s_5(\omega^3) = 4445\,, \qquad s_5(\omega^\omega) = 345\,, \qquad s_5(\omega^{\omega^\omega}) = 2345\,, \\ s_5(\omega^3 + \omega^2) = 4445445\,, \qquad s_5(\omega \cdot 3) = 454545\,. \end{gathered}$$

We may omit the subscript when $a = d$, e.g. writing $s(1) = d$.

Successors and Limits Let $x = y_d\, y_{d-1} \dots y_a\, a^\frown d$ be the decomposition of $x \in P_d \setminus \varepsilon$. By (15), x encodes a successor ordinal $\eta(x) = \beta + 1$ iff $a = d$, i.e., if x ends with two d's (or has length 1). Since then $\beta = \eta(y_d \dots y_a)$, one obtains the "predecessor of x" by removing the final d.

If $a < d$, x encodes a limit λ. Combining (11) and (15), one obtains the encoding $(x)_n$ of λ_n with

$$(x)_n = y_d\, y_{d-1} \dots y_{a+1} \big(y_a(a+1)\big)^n (a+2)^\frown d\,. \tag{16}$$

E.g., with $d = 5$, decomposing $x = 333345 = s(\omega^{\omega^4})$ gives $a = 3$, $x = y_5 y_4 y_3 3^\frown 5$, with $y_3 = 333$ and $y_5 = y_4 = \varepsilon$. Then $(x)_n = (3334)^n 5$, agreeing with, e.g. $s(\omega^{\omega^3 \cdot 2}) = 333433345$.

Robustness Translated to ordinals, Prop. 3 means that, whenever $x \leq_\# x'$ for $x, x' \in P_a$, then the corresponding ordinal $\eta(x)$ will be "structurally" smaller than $\eta(x')$. This in turn yields that the corresponding Hardy function $H^{\eta(x)}$ grows at most as fast as $H^{\eta(x')}$; see App. C for details:

Proposition 4 (Robustness)**.** *Let $a \geq 0$ and $x, x' \in P_a$. If $x \leq_\# x'$ then $H^{\eta(x)}(n) \leq H^{\eta(x')}(n')$ for all $n \leq n'$ in $\mathbb{N}$.*

6.2 Robust Hardy Computations in PCSs

Our PCSs for robust Hardy computations use three channels (see Figure 3), storing (codes for) a pair α, n on channels `o` (for "ordinal") and `c` (for "counter"), and employ an extra channel, `t`, for "temporary" storage. Instead of Σ_d, we use Σ_{d+1} with $d+1$ used as a position marker and written \$ for clarity: each channel always contains a single occurrence of \$.

`o`:	3 3 4 5 4 5 \$	the *ordinal* term $\omega^{\omega^2} + \omega^\omega$
`c`:	0 0 0 0 \$	the *counter* value 4
`t`:	\$	the *temporary* storage

Figure 3: Channels for Hardy computations.

Definition 1. A *weak Hardy computer* for Ω_{d+1} is a $(d+1)$-PCS S with channels $\mathtt{Ch} = \{\mathtt{o}, \mathtt{c}, \mathtt{t}\}$ and two distinguished states p_{beg} and p_{end} such that:

$$\begin{array}{l}\text{if } (p_{\mathrm{beg}}, x\$, y\$, z\$) \xrightarrow{*}_{\mathrm{w}} (p_{\mathrm{end}}, u, v, w) \\ \quad \text{then } x \in P_d, y \in 0^+, z = \varepsilon \text{ and } u, v, w \in \Sigma_d^*\$\,,\end{array} \qquad \text{(safety)}$$

$$\begin{array}{l}\text{if } (p_{\mathrm{beg}}, s(\alpha)\$, 0^n\$, \$) \xrightarrow{*}_{\mathrm{w}} (p_{\mathrm{end}}, s(\beta)\$, 0^m\$, \$) \\ \quad \text{then } H^\alpha(n) \geq H^\beta(m)\,.\end{array} \qquad \text{(robustness)}$$

Furthermore S is *complete* if for any $\alpha < \Omega_{d+1}$ and $n > 0$ it has runs $(p_{\mathrm{beg}}, s(\alpha)\$, 0^n\$, \$) \xrightarrow{*}_{\mathrm{w}} (p_{\mathrm{end}}, \$, 0^m\$, \$)$ where $m = H^\alpha(n)$, and it is *inv-complete* if it has runs $(p_{\mathrm{beg}}, \$, 0^m\$, \$) \xrightarrow{*}_{\mathrm{w}} (p_{\mathrm{end}}, s(\alpha)\$, 0^n\$, \$)$.

In the rest of Section 6.2 we prove the following:

Lemma 3 (PCSs weakly compute Hardy functions)**.** *For every $d \in \mathbb{N}$, there exists a weak Hardy computer S_d for Ω_{d+1} that is complete, and a weak S'_d that is inv-complete. Furthermore S_d and S'_d can be generated uniformly from d.*

We design a complete weak Hardy computer by assembling several components. Our strategy is to implement in a PCS the *canonical* Hardy steps, denoted with $\xrightarrow{H}$, and specified by the following two rewrite rules:

$$(\alpha + 1, n) \xrightarrow{H} (\alpha, n + 1) \qquad \text{for successors,} \qquad (17)$$

$$(\lambda, n) \xrightarrow{H} (\lambda_n, n) \qquad \text{for limits.} \qquad (18)$$

6.2.1 Successor Steps

We start with "canonical successor steps", as per (17). They are implemented by S_1, the PCS depicted in Figure 4. When working on codes, replacing $s(\alpha+1)$ by $s(\alpha)$ simply means removing the final d (see Section 6.1), but when the strings are in fifo channels this requires reading the whole contents of a channel and writing it back, relying on the \$ end-marker.

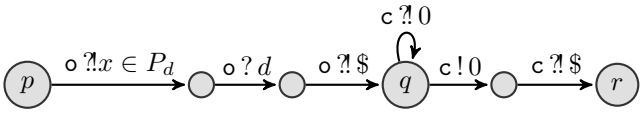

Figure 4: S_1, a PCS for Hardy steps $(\alpha + 1, n) \xrightarrow{H} (\alpha, n + 1)$.

Remark 4 (Notational/graphical conventions). The label edge "$q \xrightarrow{\mathtt{c?!0}} q$" in Figure 4, with $\mathtt{c?!0}$ as label, is shorthand notation for "$q \xrightarrow{\mathtt{c?0}} \circ \xrightarrow{\mathtt{c!0}} q$", letting the intermediary state remain implicit. We also use meta-rules like "$p \xrightarrow{\mathtt{o?!}\,x \in P_d} \circ$" above to denote a subsystem tasked with reading and writing back a string x over $\mathtt{o}$ while checking that it belongs to P_d; since P_d is a regular language, such subsystems are trivial to implement.

We first analyze the behavior of S_1 when superseding of low-priority messages does not occur, i.e., we first consider its "reliable" semantics. In this case,

starting S_1 in state p performs the step given in (17) for successor ordinals. More precisely, S_1 guarantees

$$\begin{array}{rl} & (p,\ s(\alpha+1)\$,\ 0^n\$,\ \$) \xrightarrow{*}_{\mathrm{rel}} (r,u,v,w) \\ \text{iff} & u = s(\alpha)\$ \ \wedge\ v = 0^{n+1}\$ \ \wedge\ w = \$\,. \end{array} \tag{19}$$

Note that (19) refers to "$\xrightarrow{*}_{\mathrm{rel}}$", with no superseding.

Observe that S_1 has to non-deterministically guess where the end of $s(\alpha)$ occurs before reading $d\$$ in channel $\mathtt{o}$, and will deadlock if it guesses incorrectly. We often rely on this kind of non-deterministic programming to reduce the size of the PCSs we build. Finally, we observe that if x does not end with dd (and is not just d), i.e., if $\eta(x)$ is not a successor ordinal, then S_1 will certainly deadlock.

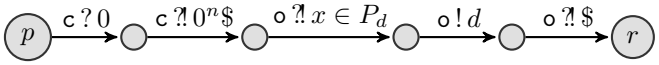

Figure 5: S_2, a PCS for inverse Hardy steps $(\alpha, n+1) \xrightarrow{H^{-1}} (\alpha+1, n)$.

We now consider S_2, the PCS depicted in Figure 5 that implements the inverse canonical steps $(\alpha, n+1) \xrightarrow{H^{-1}} (\alpha+1, n)$. Implementing such steps on codes is an easy string-rewriting task since $s(\alpha+1) = s(\alpha)d$, however our PCS must again read the whole contents of its channels, write them back with only minor modifications while fulfilling the safety requirement of Def. 1. When considering the reliable behavior, S_2 guarantees

$$\begin{array}{l} (p,\ x\$,\ y\$,\ \$) \xrightarrow{*}_{\mathrm{rel}} (r,u,v,w) \\ \text{iff} \left\{ \begin{array}{l} x \in P_d, y = 0^{n+1} \text{ for some } n, \\ u = s(\eta(x)+1)\$, v = 0^n\$, \text{ and } w = \$\,. \end{array} \right. \end{array} \tag{20}$$

Consider now the behavior of S_1 *when superseding may occur*. Note that a run $(p, x\$, y\$, z\$) \xrightarrow{*}_{\mathrm{w}} (r, \ldots)$ from p to r is a *single-pass* run: it reads the whole contents of channels $\mathtt{o}$ and $\mathtt{c}$ once, and writes some new contents. This feature assumes that we start with a single \$ at the end of each channel, as expected by S_1. For such single-pass runs, the PCS behavior with superseding semantics can be derived from the reliable behavior: for single-pass runs, $C \xrightarrow{*}_{\mathrm{w}} D$ iff $C \xrightarrow{*}_{\mathrm{rel}} D' \geq_{\#} D$ for some D'.

Combined with (19), the above remark entails robustness for S_1: $(p, s(\alpha+1)\$, 0^n\$, \$) \xrightarrow{*}_{\mathrm{w}} (r, s(\beta)\$, 0^{n'}\$, \$)$ iff $s(\beta) \leq_{\#} s(\alpha)$ and $0^{n'}\$ \leq_{\#} 0^{n+1}\$$, i.e., $n' \leq n+1$. With Prop. 4, we deduce $H^\beta(n') \leq H^\alpha(n)$.

The same reasoning applies to S_2 since this PCS also performs single-pass runs from p to r, hence $(p, s(\alpha)\$, 0^n\$, \$) \xrightarrow{*}_{\mathrm{w}} (r, s(\beta)\$, 0^{n'}\$, \$)$ iff $s(\beta) \leq_{\#} s(\alpha+1)$ and $n' \leq n-1$. Thus $H^\beta(n') \leq H^\alpha(n)$.

6.2.2 Limit Steps

Our next component is S_3, see Figure 6, that implements the canonical Hardy steps for limits from (18). The construction follows (16): $S_{3,a}$ reads (and writes back) the contents of channel $\mathtt{o}$, guessing non-deterministically the decomposition $y_d \ldots y_{a+1} y_a a(a+1)^\frown d$ of $s(\lambda)$, it writes back $y_d \ldots y_{a+1}$ and copies y_a on the temporary $\mathtt{t}$ with $a+1$ appended. Then, a loop around state q_a copies

0^n from and back to $\mathtt{c}$. Every time one 0 is transferred, the whole contents of $\mathtt{t}$, initialized with $y_a(a+1)$, is copied to $\mathtt{o}$. When the loop has been visited n times, $S_{3,a}$ empties $\mathtt{t}$ and resumes the transfer of $s(\lambda)$ by copying the final $(a+2)^\frown d$.

For clarity, $S_{3,a}$ as given in Figure 6 assumes that a is fixed. The actual S_3 component guesses non-deterministically what is the value of a for the $s(\lambda)$ code on $\mathtt{o}$ and gives the control to $S_{3,a}$ accordingly.

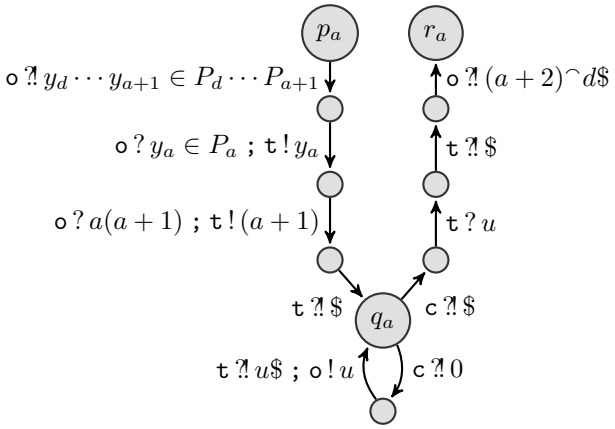

Figure 6: $S_{3,a}$, a PCS for Hardy steps $(\lambda, n) \xrightarrow{H} (\lambda_n, n)$.

As far as reliable steps are considered, S_3 guarantees

$$\begin{array}{c}(p,\ s(\alpha)\$,\ 0^n\$,\ \$) \xrightarrow{*}_{\mathrm{rel}} (r, u, v, w)\\ \text{iff } \alpha \in \mathit{Lim}, u = s(\alpha_n)\$, v = 0^n\$, \text{ and } w = \$\,.\end{array} \tag{21}$$

If superseding is allowed, a run $(p_a, s(\alpha)\$, 0^n\$, \$) \xrightarrow{*}_{\mathrm{w}} (r_a, u, v, w)$ has the form

$$\begin{aligned}&(p_a, s(\alpha)\$, 0^n\$, \$) \xrightarrow{*}_{\mathrm{w}} C_0 = (q_a, (a+2)^\frown d\$x_0, 0^n\$, z_0\$)\\ &\xrightarrow{*}_{\mathrm{w}} C_1 = (q_a, (a+2)^\frown d\$x_1, 0^{n-1}\$v_1, z_1\$) \xrightarrow{*}_{\mathrm{w}} \cdots\\ &\xrightarrow{*}_{\mathrm{w}} C_n = (q_a, (a+2)^\frown d\$x_n, \$v_n, z_n\$) \xrightarrow{*}_{\mathrm{w}} (r_a, x'_n\$, v'_n\$, \$)\end{aligned}$$

where $C_i = (q_a, (a+2)^\frown d\$x_i, 0^{n-i}\$v_i, z_i\$)$ occurs when state q_a is visited for the i-th time. Since the run is single-pass on $\mathtt{c}$, we know that $v_i \leq_\# 0^i$ for all $i = 0, \ldots, n$. Since it is single-pass on $\mathtt{o}$, we deduce that $x_0 \leq_\# y_d \ldots y_{a+1}$, then $x_{i+1} \leq_\# x_i z_i$ for all i, and finally $x'_n \leq_\# x_n(a+2)^\frown d$, with also $z_0 \leq_\# y_a(a+1)$. Finally, $z_{i+1} \leq_\# z_i$ since each subrun $C_i \xrightarrow{*}_{\mathrm{w}} C_{i+1}$ is single-pass on $\mathtt{t}$.

All this yields $x'_n \leq_\# s(\lambda_n)$ and $v'_n \leq_\# 0^n$. Hence S_3 is safe and robust: $(p, s(\alpha)\$, 0^n\$, \$) \xrightarrow{*}_{\mathrm{w}} (r, s(\beta), 0^{n'}\$, \$)$ iff $\alpha \in \mathit{Lim}$, $s(\beta) \leq_\# s(\alpha_n)$ and $n' \leq n$, entailing $H^\beta(n') \leq H^\alpha(n)$.

There remains to consider S_4, see Figure 7, the PCS component that implements inverse Hardy steps for limits. For given $a < d$, $S_{4,a}$ assumes that channel $\mathtt{o}$ contains $s(\lambda_n) = y_d \ldots y_{a+1}[y_a(a+1)]^n(a+2)^\frown d$, guesses the position of the first $y_a(a+1)$ factor, and checks that it indeed occurs n times if $\mathtt{c}$ contains 0^n. This check uses copies $z_1, z_2, \ldots$ of $y_a(a+1)$ temporarily stored on $\mathtt{t}$. Then S_4

writes back $s(\lambda) = y_d \dots y_{a+1} z a^\frown d$ on $\mathtt{o}$, where $z(a+1) = z_n$. The reader should be easily convinced that, as far as one considers reliable steps, S_4 guarantees

$$\begin{array}{c}(p,\ s(\alpha)\$,\ 0^n\$,\ \$) \xrightarrow{*}_{\mathrm{rel}} (r, u, v, w) \text{ iff} \\ \exists \lambda \in Lim : \alpha = \lambda_n, u = s(\lambda)\$, v = 0^n\$, \text{ and } w = \$\,.\end{array} \tag{22}$$

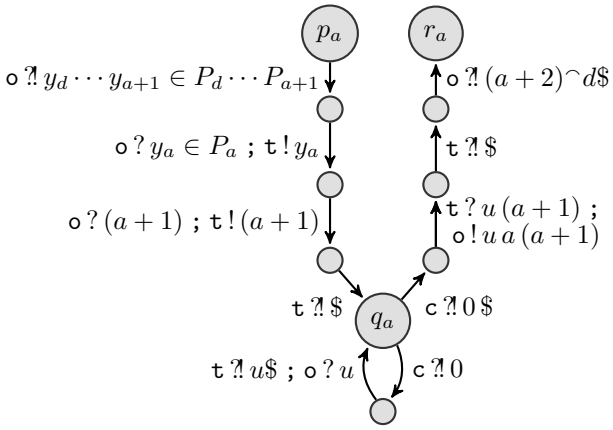

Figure 7: $S_{4,a}$, a PCS for inverse Hardy steps $(\lambda_n, n) \xrightarrow{H^{-1}} (\lambda, n)$.

When superseding is taken into account, a run from p to r in S_4 has the form $(p, s(\alpha)\$, 0^n\$, \$) \xrightarrow{*}_{\mathrm{w}} C_1 \xrightarrow{*}_{\mathrm{w}} C_2 \xrightarrow{*}_{\mathrm{w}} \cdots C_n \xrightarrow{*}_{\mathrm{w}} (r, u, v, w)$ where, for $i = 1, \dots, n$, C_i is the i-th configuration that visits state q_a. Necessarily, C_i is some $(q_a, x_i\$x, 0^{n-i}0\$v_i, z_i\$)$. The first visit to q_a has $x \leq_\# y_d \dots y_{a+1}$, $z_1 \leq_\# y_a(a+1)$ and $v_1 = \varepsilon$, the following ones ensure $x_i = z_i x_{i+1}$, $z_{i+1} \leq_\# z_i$ and $v_{i+1} \leq_\# v_i 0$. Concluding the run requires $x_n = (a+2)^\frown d$. Finally $v \leq_\# 0^n\$$, $s(\beta) = y_d \dots y_{a+1}(a+1) z_1 \dots z_{n-1}(a+2)^\frown d$ and $u \leq_\# y_d \dots y_{a+1} z a^\frown d$ for $z(a+1) = z_n \leq_\# z_{n-1} \leq_\# \cdots z_2 \leq_\# z_1 \leq_\# y_a(a+1)$. Thus $u = s(\beta)\$$ and $v = 0^{n'}\$$ imply $s(\beta) \leq_\# s(\lambda)$ for some λ with $s(\lambda_n) \leq_\# s(\alpha)$, yielding $H^\beta(n') \leq H^\lambda(n) = H^{\lambda_n}(n) \leq H^\alpha(n)$.

6.3 Wrapping It Up

With the above weak Hardy computers, we have the essential gadgets required for our reductions. The wrapping-up is exactly as in (Haddad et al., 2012; Schnoebelen, 2010a) (with a different encoding and a different machine model) and will only be summarily explained.

Theorem 5 (Verifying PCSs is Hard)**.** *Reachability and Termination of PCSs are* $\mathbf{F}_{\varepsilon_0}$*-hard.*

Proof. We exhibit a LogSpace reduction from the halting problem of a Turing machine M working in F_{ε_0} space to the Reachability problem in a PCS. We assume wlog. M to start in a state p_0 with an empty tape and to have a single halting state p_h that can only be reached after clearing the tape.

Figure 8 depicts the PCS S we construct for the reduction. Let $n \stackrel{\text{def}}{=} |M|$ and $d \stackrel{\text{def}}{=} n + 1$. A run in S from the initial configuration to the final one goes through three stages:

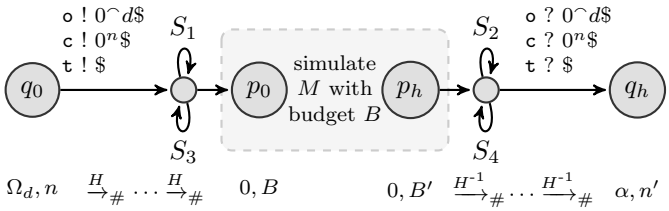

Figure 8: Schematics for Thm. 5.

1. The first stage robustly computes $F_{\varepsilon_0}(|M|) = H^{\Omega_d}(n)$ by first writing $s(\Omega_d)\$ = 0^\frown d\$$ on $\mathtt{o}$, $0^n\$$ on $\mathtt{c}$, and $\$$ on $\mathtt{t}$, then by using S_1 and S_3 to perform forward Hardy steps; thus upon reaching state p_0, $\mathtt{o}$ and $\mathtt{t}$ contain $\$$ and $\mathtt{c}$ encodes a budget $B \leq F_{\varepsilon_0}(|M|)$.

2. The central component simulates M over $\mathtt{c}$ where the symbols 0 act as blanks—this is easily done by cycling through the channel contents to simulate the moves of the head of M on its tape. Due to superseding steps, the outcome of this phase upon reaching p_h is that $\mathtt{c}$ contains $B' \leq B$ symbols 0.

3. The last stage robustly computes $(F_{\varepsilon_0})^{-1}(B')$ by running S_2 and S_4 to perform backward Hardy steps. This leads to $\mathtt{o}$ containing the encoding of some ordinal α and $\mathtt{c}$ of some n', but we empty these channels and check that $\alpha = \Omega_d$ and $n' = n$ before entering state q_h.

Because

$$H^{\Omega_d}(n) \geq B \geq B' \geq H^{\alpha}(n') = H^{\Omega_d}(n)\,, \tag{23}$$

all the inequalities are actually equalities, and the simulation of M in stage 2 has necessarily employed reliable steps. Therefore, M halts if and only if $(q_h, \varepsilon, \varepsilon, \varepsilon)$ is reachable from $(q_0, \varepsilon, \varepsilon, \varepsilon)$ in S.

The case of (non-)Termination is similar, but employs a *time* budget in a separate channel in addition to the space budget, in order to make sure that the simulation of M terminates in all cases, and leads to a state q_h that is the only one from which an infinite run can start in S. □

7 Concluding Remarks

We introduced Priority Channel Systems, a natural model for protocols and programs with differentiated, prioritized asynchronous communications, and showed how they give rise to well-structured systems with decidable model-checking problems.

We showed that Reachability and Termination for PCSs are $\mathbf{F}_{\varepsilon_0}$-complete, and we expect our techniques to be transferable to other models, e.g. models based on wqos on bounded-depth trees or graphs, whose complexity has not been analyzed (Genest et al., 2008; Delzanno et al., 2010; Meyer, 2008; Bansal et al., 2013). This is part of our current research agenda on complexity for well-structured systems (Schmitz and Schnoebelen, 2011).

In spite of their enormous worst-case complexity, we expect PCSs to be amenable to regular model checking techniques *à la* (Abdulla and Jonsson, 1996;

Boigelot and Godefroid, 1999). This requires investigating the algorithmics of upward- and downward-closed sets of configurations wrt. the priority ordering. These sets, which are always regular, seem promising since $\sqsubseteq_{\mathrm{p}}$ shares some good properties with the better-known subword ordering, e.g. the upward- or downward-closure of a sequence $x \in \Sigma_d^*$ can be represented by a DFA with $|x|$ states.

Acknowledgments

We thank Lev Beklemishev who drew our attention to (Schütte and Simpson, 1985).

References

Abdulla, P.A. and Jonsson, B., 1996. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101. doi:10.1006/inco.1996.0053.

Abdulla, P.A., Čerāns, K., Jonsson, B., and Tsay, Y.K., 2000. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160 (1–2):109–127. doi:10.1006/inco.1999.2843.

Abdulla, P.A., Deneux, J., Ouaknine, J., and Worrell, J., 2005. Decidability and complexity results for timed automata via channel machines. In *ICALP 2005*, *32nd International Colloquium on Automata, Languages and Programming*, volume 3580 of *Lecture Notes in Computer Science*, pages 1089–1101. Springer. doi:10.1007/11523468_88.

Abdulla, P.A., Atig, M.F., and Cederberg, J., 2012. Timed lossy channel systems. In *FST&TCS 2012*, *32nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 18 of *Leibniz International Proceedings in Informatics*, pages 374–386. Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSTTCS.2012.374.

Bansal, K., Koskinen, E., Wies, T., and Zufferey, D., 2013. Structural counter abstraction. In Piterman, N. and Smolka, S., editors, *TACAS 2013*, *19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science. Springer. To appear.

Boigelot, B. and Godefroid, P., 1999. Symbolic verification of communication protocols with infinite state spaces using QDDs. *Formal Methods in System Design*, 14(3): 237–255. doi:10.1023/A:1008719024240.

Bouajjani, A. and Habermehl, P., 1999. Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. *Theoretical Computer Science*, 221(1–2):211–250. doi:10.1016/S0304-3975(99)00033-X.

Bouyer, P., Markey, N., Ouaknine, J., Schnoebelen, Ph., and Worrell, J., 2012. On termination and invariance for faulty channel machines. *Formal Aspects of Computing*, 24(4–6):595–607. doi:10.1007/s00165-012-0234-7.

Cécé, G., Finkel, A., and Purushothaman Iyer, S., 1996. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1):20–31. doi:10.1006/inco.1996.0003.

Cécé, G. and Finkel, A., 2005. Verification of programs with half-duplex communication. *Information and Computation*, 202(2):166–190. doi:10.1016/j.ic.2005.05.006.

Chambart, P. and Schnoebelen, Ph., 2008. The ordinal recursive complexity of lossy channel systems. In *LICS 2008*, *25th Annual IEEE Symposium on Logic in Computer Science*, pages 205–216. IEEE Press. doi:10.1109/LICS.2008.47.

Delzanno, G., Sangnier, A., and Zavattaro, G., 2010. Parameterized verification of ad hoc networks. In Gastin, P. and Laroussinie, F., editors, *Concur 2010*, *21st International Conference on Concurrency Theory*, volume 6269 of *Lecture Notes in Computer Science*, pages 313–327. Springer. doi:10.1007/978-3-642-15375-4_22.

Fairtlough, M. and Wainer, S.S., 1998. Hierarchies of provably recursive functions. In Buss, S., editor, *Handbook of Proof Theory*, chapter III, pages 149–207. Elsevier. doi:10.1016/S0049-237X(98)80018-9.

Finkel, A. and Schnoebelen, Ph., 2001. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92. doi:10.1016/S0304-3975(00)00102-X.

Genest, B., Muscholl, A., Serre, O., and Zeitoun, M., 2008. Tree pattern rewriting systems. In Cha, S., Choi, J.Y., Kim, M., Lee, I., and Viswanathan, M., editors, *ATVA 2008*, *6th International Symposium on Automated Technology for Verification and Analysis*, volume 5311 of *Lecture Notes in Computer Science*, pages 332–346. Springer. doi:10.1007/978-3-540-88387-6_29.

Gupta, A., 1992. A constructive proof that trees are well-quasi-ordered under minors. In Nerode, A. and Taitslin, M., editors, *LFCS 1992*, *2nd International Symposium on Logical Foundations of Computer Science*, volume 620 of *Lecture Notes in Computer Science*, pages 174–185. Springer. doi:10.1007/BFb0023872.

Haddad, S., Schmitz, S., and Schnoebelen, Ph., 2012. The ordinal-recursive complexity of timed-arc Petri nets, data nets, and other enriched nets. In *LICS 2012*, *27th Annual IEEE Symposium on Logic in Computer Science*, pages 355–364. IEEE Press. doi:10.1109/LICS.2012.46.

Kurucz, A., 2006. Combining modal logics. In *Handbook of Modal Logics*, chapter 15, pages 869–926. Elsevier. doi:10.1016/S1570-2464(07)80018-8.

Lasota, S. and Walukiewicz, I., 2008. Alternating timed automata. *ACM Transactions on Computational Logic*, 9(2). doi:10.1145/1342991.1342994.

Löb, M. and Wainer, S., 1970. Hierarchies of number theoretic functions, I. *Archiv für Mathematische Logik und Grundlagenforschung*, 13:39–51. doi:10.1007/BF01967649.

Meyer, R., 2008. On boundedness in depth in the π-calculus. In Ausiello, G., Karhumäki, J., Mauri, G., and Ong, L., editors, *IFIP TCS 2008*, *Fifth IFIP International Conference on Theoretical Computer Science*, volume 273 of *IFIP*, pages 477–489. Springer. doi:10.1007/978-0-387-09680-3_32.

Ossona de Mendez, P. and Nešetřil, J., 2012. *Sparsity*, chapter 6: Bounded height trees and tree-depth, pages 115–144. Springer. doi:10.1007/978-3-642-27875-4_6.

Ouaknine, J. and Worrell, J., 2007. On the decidability and complexity of Metric Temporal Logic over finite words. *Logical Methods in Computer Science*, 3(1):1–27. doi:10.2168/LMCS-3(1:8)2007.

Schmitz, S. and Schnoebelen, Ph., 2011. Multiply-recursive upper bounds with Higman's lemma. In *ICALP 2011*, *38th International Colloquium on Automata, Languages and Programming*, volume 6756 of *Lecture Notes in Computer Science*, pages 441–452. Springer. doi:10.1007/978-3-642-22012-8_35.

Schmitz, S. and Schnoebelen, Ph., 2012. Algorithmic aspects of WQO theory. Lecture notes. http://cel.archives-ouvertes.fr/cel-00727025.

Schnoebelen, Ph., 2010a. Revisiting Ackermann-hardness for lossy counter machines and reset Petri nets. In Hliněný, P. and Kučera, A., editors, *MFCS 2010*, *35th International Symposium on Mathematical Foundations of Computer Science*, volume 6281 of *Lecture Notes in Computer Science*, pages 616–628. Springer. doi:10.1007/978-3-642-15155-2_54.

Schnoebelen, Ph., 2010b. Lossy counter machines decidability cheat sheet. In Kučera,

A. and Potapov, I., editors, *RP 2010*, *4th Workshop on Reachability Problems*, volume 6227 of *Lecture Notes in Computer Science*, pages 51–75. Springer. doi: 10.1007/978-3-642-15349-5_4.

Schütte, K. and Simpson, S.G., 1985. Ein in der reinen Zahlentheorie unbeweisbarer Satz über endliche Folgen von natürlichen Zahlen. *Archiv für Mathematische Logik und Grundlagenforschung*, 25(1):75–89. doi:10.1007/BF02007558.

A Proof of Prop. 1

Let $S = (\Sigma_d, \mathtt{Ch}, Q, \Delta)$ be a d-PCS. Recall that its behavior under write-superseding policy is given by $\mathcal{S}_{\mathrm{w}} = (\mathit{Conf}_S, \to_{\mathrm{w}})$, while its behavior under internal-superseding policy is given by $\mathcal{S}_{\#} \stackrel{\text{def}}{=} (\mathit{Conf}_S, \to_{\#})$.

Lemma 4 (From $\mathcal{S}_{\mathrm{w}}$ to $\mathcal{S}_{\#}$)**.** *If $\mathcal{S}_w$ has a run $C \stackrel{*}{\to}_w D$ then $\mathcal{S}_{\#}$ has a run $C \stackrel{*}{\to}_{\#} D$.*

Proof. We show that $\to_{\mathrm{w}}$ is contained in $\stackrel{+}{\to}_{\#}$, assuming for the sake of simplicity that S has only one channel.

A writing step $(p,x) \stackrel{!a}{\to}_{\mathrm{w}} (q,y)$ with $x = zb_1 \dots b_j$ and $y = za$ in $\mathcal{S}_{\mathrm{w}}$ can be simulated in $\mathcal{S}_{\#}$ with $(p,x) \stackrel{!a}{\to}_{\#} (q, zb_1 \dots b_j a) \stackrel{\#\ell}{\to}_{\#} (q, zb_1 b_{j-1}) \stackrel{\#\ell-1}{\longrightarrow}_{\#} \cdots \stackrel{\#k+1}{\longrightarrow}_{\#} (q, za)$, where $\ell = |x|$ and $k = |z|$. Reading steps simply coincide in $\mathcal{S}_{\mathrm{w}}$ and $\mathcal{S}_{\#}$. □

In the other direction, one can translate runs in $\mathcal{S}_{\#}$ to runs in $\mathcal{S}_{\mathrm{w}}$ as stated by following lemma.

Lemma 5 (From $\mathcal{S}_{\#}$ to $\mathcal{S}_{\mathrm{w}}$)**.** *If $\mathcal{S}_{\#}$ has a run $C \stackrel{*}{\to}_{\#} D$ then $\mathcal{S}_w$ has a run $C' \stackrel{*}{\to}_w D$ for some $C' \leq_{\#} C$.*
(In particular, if the channels are empty in C, then necessarily $C' = C$ and $C \stackrel{*}{\to}_w D$.)

Proof. Again we assume that S has only one channel.

Write the run $C \stackrel{*}{\to}_{\#} D$ under the form $C_0 \to_{\#} C_1 \to_{\#} \cdots \to_{\#} C_n$ and rearrange its steps so that superseding occurs greedily. This relies on Lemma 6 stated just below.

Repeatedly applying Lemma 6 to transform $C_0 \stackrel{*}{\to}_{\#} C_n$ as long as possible is bound to terminate (with each commutation, superseding steps are shifted to the left of reliable steps, or the sum $\sum_i k_i$ of superseding positions in steps $C_{i-1} \stackrel{\#k_i}{\longrightarrow}_{\#} C_i$ increases strictly while being bounded by $O(n^2)$ for a length-n run). One eventually obtains a new run $C_0 \stackrel{*}{\to}_{\#} C_n$ with same starting and final configurations, and where all the superseding steps occur (at the beginning of the run or) just after a write in *normalized* sequences of the form

$$C = (q,x) \stackrel{!a}{\to}_{\#} \stackrel{\#\ell}{\to}_{\#} \stackrel{\#\ell-1}{\longrightarrow}_{\#} \stackrel{\#\ell-2}{\longrightarrow}_{\#} \cdots \stackrel{\#\ell-r}{\longrightarrow}_{\#} C' \,, \tag{24}$$

where furthermore $\ell = |x|$. In this case, $\mathcal{S}_{\mathrm{w}}$ has a step $C \stackrel{!a}{\to}_{\mathrm{w}} C'$.

Greedily shifting superseding steps to the left may move some of them at the start of the run instead of after a write: these steps are translated into $C \geq_{\#} C'$ in Lemma 5. Finally, the steps that are not in normalized sequences are reading steps which exist unchanged in $\mathcal{S}_{\mathrm{w}}$. □

Lemma 6 (Commuting #-steps)**.**

1. *If $C_1 \stackrel{?a}{\to}_{\#} C_2 \stackrel{\#k}{\to}_{\#} C_3$ then there is a configuration C_2' s.t. $C_1 \stackrel{\#k+1}{\longrightarrow}_{\#} C_2' \stackrel{?a}{\to}_{\#} C''$.*

2. $C_1 = (q, x) \xrightarrow{!a}_{\#} C_2 \xrightarrow{\#k}_{\#} C_3$ *with* $k < |x|$, *then there is a configuration* C_2' *s.t.* $C_1 \xrightarrow{\#k}_{\#} C_2' \xrightarrow{!a}_{\#} C_3$.

3. *If* $C_1 = (q, x) \xrightarrow{\#k_1}_{\#} C_2 \xrightarrow{\#k_2}_{\#} C_3$ *with* $k_1 \leq k_2$ *then there is a configuration* C_2' *s.t.* $C_1 \xrightarrow{\#k_2+1}_{\#} C_2' \xrightarrow{\#k_1}_{\#} C''$.

B PCSs and LCSs

It is easy to see that Priority Channel Systems are at least as expressive as Lossy Channel Systems, and even the Dynamic Lossy Channel Systems (DLCS) recently introduced by Abdulla et al. (2012).

Furthermore, if we adopt the strict superseding policy described in Remark 1, PCSs can even simulate reliable channel systems, a Turing-powerful model. Since the two simulations are very similar, we start our presentation with the simpler one.

B.1 Simulating Reliable Channels by "Strict Superseding" PCSs

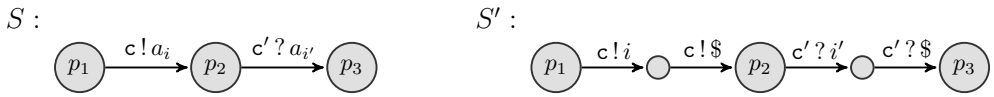

Figure 9: Simulating reliable channels with "strict" PCSs.

A channel system S with reliable channels uses a finite (un-prioritized) alphabet $\Sigma = \{a_0, \ldots, a_{p-1}\}$ and is equipped with m "standard" channels $\mathtt{c}_1, \ldots, \mathtt{c}_m$. We simulate S with a PCS S' having the same m channels and using the Σ_d priority alphabet with $d = p$. We use $d \in \Sigma_d$ as a separator, denoted \$ for clarity, while the other priorities $i = 0, \ldots, p-1$ represent the original messages a_i. A string $w = a_{i_1} \ldots a_{i_n}$ in Σ^* will be encoded as $\widetilde{w} \stackrel{\text{def}}{=} i_1 \,\$ \ldots i_n \,\$ \in \Sigma_d^*$ when in S', see Figure 9 for the construction.

With the *strict superseding* policy, the only superseding that can occur is to have \$ overtake and erase a preceding $i <$ \$. This results in a channel containing two (or more) consecutive \$, a pattern that can never disappear in this simulation and that eventually forbid reading on the involved channel. In particular, any run of S' that reaches $C_{\text{end}} = (q_{\text{end}}, \varepsilon, \ldots, \varepsilon)$ has not used any (strict) superseding and thus corresponds to a run of S.

With this reduction one sees that reachability is undecidable for PCSs with the strict superseding policy considered in Remark 1.

B.2 Simulation Dynamic LCSs by PCSs

A DLCS S has $\Sigma = \{a_0, \ldots, a_{p-1}\}$ and $\mathtt{Ch} = \{\mathtt{c}_1, \ldots, \mathtt{c}_m\}$ as above. It also has a *second-order channel* $\mathtt{c}_0$ that is a fifo buffer of *sequences* over Σ, i.e. of channel contents. Transition rules may read and write from standard channels with the usual "$\mathtt{c}_j!a_i$" and "$\mathtt{c}_j?a_i$" operations. Rules may also append a complete copy of a channel contents to the second-order channel with a "$!!\mathtt{c}_j$" operation, or read

a channel contents from c_0 with "$??\mathsf{c}_j$": this replaces the contents of c_j with the sequence read from c_0. On top of this behavior, the system is unreliable, "lossy", and messages inside the channels may be lost nondeterministically. Inside the second-order c_0, whole sequences may be lost as well as individual messages inside sequences. These are two different losing modalities, in the first case c_0 ends up containing less sequences, in the second case it ends up with shorter sequences.

Our simulation of a DLCS S with a PCS S' uses $m+1$ channels and a priority alphabet Σ_d having level $d = p + 1$. Since in this simulation $p, p + 1 \in \Sigma_d$ are used as markers, we denote them with the special symbols \$ and £. Hence $£ > \$ > i$ for $i = 0, \ldots, p - 1$. A sequence $w \in \Sigma^*$ is be represented by $\widetilde{w}$ as in the previous simulation. A sequence $w_1, \ldots, w_n$ stored in the second-order channel of S will be represented by $\widetilde{w_1}£ \ldots \widetilde{w_n}£$ in S': see Figure 10.

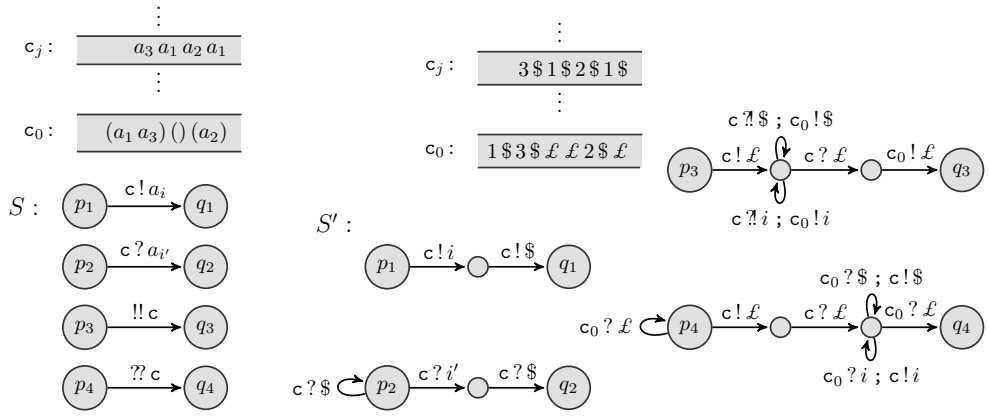

Figure 10: Simulating DLCSs with PCSs.

When defining S' graphically, we use some shorthand notation (e.g. "$\mathsf{c}??x$" to read and write back a symbol x in a single step) explained in Remark 4. Going further, *higher-order* lossy channel systems—DLCSs being "second-order LCSs"—can also be simulated by suitably adding new high-priority separators.

One can tighten these simulations to use fewer priorities by encoding the messages $a_0, \ldots, a_{p-1}$ as *fixed length* binary strings over $\{0, 1\}$ followed by a \$ separator. Then the prioritized alphabet $\{0, 1, \$, £\}$ suffices, and $\{0, 1, \$\}$ is enough for LCS. In the case of *weak* LCSs where the set of messages is linearly ordered (say $a_0 < a_1 < \cdots < a_{p-1}$) and where, in addition to message losses, any message can be replaced by a lower message inside the channels, we can further tighten this to $\{0, \$\}$ with a unary encoding of message a_i as $0^i\$$.

Since this simulation preserves reachability (modulo the encoding of configurations) and termination, we conclude that verifying safety and inevitability properties of PCSs must be at least as hard as it is for LCSs, i.e., $\mathbf{F}_{\omega^\omega}$-hard (Chambart and Schnoebelen, 2008; Schmitz and Schnoebelen, 2011). We also conclude that repeated control-state reachability (and several other problems, see (Schnoebelen, 2010a)) are undecidable for PCSs since they are undecidable for LCSs.

C Hardy Computations

We provide in this section some useful properties of the Hardy computations (see (Fairtlough and Wainer, 1998), (Schmitz and Schnoebelen, 2011, App. C), or (Schmitz and Schnoebelen, 2012, App. A) for details).

C.1 Properties of the Hardy Hierarchy

The first fact is that each Hardy function is expansive and monotone in its argument n:

Fact 1 (Expansiveness and Monotonicity, see e.g. 24, Lem. C.9 and C.10)**.** *For all α, α' in Ω and $n > 0, m$ in $\mathbb{N}$,*

$$n \leq H^\alpha(n)\,, \tag{25}$$

$$n \leq m \text{ implies } H^\alpha(n) \leq H^\alpha(m)\,. \tag{26}$$

However, the Hardy functions are not monotone in the ordinal parameter: $H^{n+1}(n) = 2n+1 > 2n = H^n(n) = H^\omega(n)$, though $n+1 < \omega$. We will introduce an ordering on ordinal terms in Section C.2 that ensures monotonicity of the Hardy functions.

Another handful fact is that we can decompose Hardy computations:

Fact 2 (see e.g. 24, Lem. C.7)**.** *For all α, γ in Ω, and n in $\mathbb{N}$,*

$$H^{\gamma+\alpha}(n) = H^\gamma(H^\alpha(n))\,. \tag{27}$$

Note that (27) holds for all ordinal terms, and not only for those α, γ such that $\gamma+\alpha$ is in CNF—this is a virtue of working with terms rather than set-theoretic ordinals.

C.2 Ordinal Embedding

We introduce a partial ordering $\sqsubseteq_o$ on ordinal terms, called *embedding*, and which corresponds to the strict tree embedding on the structure of ordinal terms. Formally, it is defined by $\alpha \sqsubseteq_o \beta$ if and only if $\alpha = \omega^{\alpha_1} + \cdots + \omega^{\alpha_p}$, $\beta = \omega^{\beta_1} + \cdots + \omega^{\beta_m}$, and there exist $i_1 < i_2 < \ldots < i_p$ such that $\alpha_1 \sqsubseteq_o \beta_{i_1} \wedge \cdots \wedge \alpha_p \sqsubseteq_o \beta_{i_p}$. Note that $0 \sqsubseteq_o \alpha$ for all α, that $1 \sqsubseteq_o \alpha$ for all $\alpha > 0$. In general, $\alpha \not\sqsubseteq_o \omega^\alpha$ and $\lambda_n \not\sqsubseteq_o \lambda$. This ordering is congruent for addition and ω-exponentiation of terms:

$$\alpha \sqsubseteq_o \alpha' \text{ and } \beta \sqsubseteq_o \beta' \text{ imply } \alpha + \beta \sqsubseteq_o \alpha' + \beta'\,, \tag{28}$$

$$\alpha \sqsubseteq_o \alpha' \text{ implies } \omega^\alpha \sqsubseteq_o \omega^{\alpha'}\,, \tag{29}$$

and could in fact be defined alternatively by the axiom $0 \sqsubseteq_o \alpha$ and the two deduction rules (28) and (29).

We list a few useful consequences of the definition of $\sqsubseteq_o$:

$$\alpha \sqsubseteq_o \gamma + \omega^\beta \text{ implies } \alpha \sqsubseteq_o \gamma, \text{ or } \alpha = \gamma' + \omega^{\beta'} \text{ with } \gamma' \sqsubseteq_o \gamma \text{ and } \beta' \sqsubseteq_o \beta\,, \tag{30}$$

$$n \leq m \text{ implies } \lambda_n \sqsubseteq_o \lambda_m\,, \tag{31}$$

$$\alpha \sqsubseteq_o \lambda \text{ implies } \alpha \sqsubseteq_o \lambda_n, \text{ or } \alpha \text{ is a limit and } \alpha_n \sqsubseteq_o \lambda_n\,. \tag{32}$$

Proof of (30). Intuitively, there are two cases when we consider $\alpha \sqsubseteq_o \alpha' = \gamma + \omega^\beta$: either the ω^β summand of α' is in the range of the embedding or not. If it is not, then already $\alpha \sqsubseteq_o \gamma$. If it is, then α must be some $\gamma' + \omega^{\beta'}$ and $\omega^{\beta'} \sqsubseteq_o \omega^\beta$, which implies in turn $\beta' \sqsubseteq_o \beta$. □

Proof of (31). By induction on λ: indeed if $\lambda = \gamma + \omega^{\beta+1}$ then $\lambda_m = \gamma + \omega^\beta \cdot m$, which is $\lambda_n + \omega^\beta \cdot (m-n)$. If $\lambda = \gamma + \omega^{\lambda'}$, the ind. hyp. gives $\lambda'_n \sqsubseteq_o \lambda'_m$, hence $\lambda_n = \gamma + \omega^{\lambda'_n} \sqsubseteq_o \gamma + \omega^{\lambda'_m} = \lambda_m$. □

Proof of (32). By induction on λ. We can write λ as some $\gamma + \omega^\beta$ with $\beta > 0$ so that $\lambda_n = \gamma + (\omega^\beta)_n$. If $\alpha \sqsubseteq_o \gamma$, then $\alpha \sqsubseteq_o \lambda_n$ trivially. If $\alpha = \gamma' + 1$ is a successor, $1 \sqsubseteq_o (\omega^\beta)_n$ and again $\alpha \sqsubseteq_o \lambda_n$. There remains the case where $\alpha = \gamma' + \omega^{\beta'}$ is a limit (i.e. $\beta' > 0$) with $\gamma' \sqsubseteq_o \gamma$ and $\beta' \sqsubseteq_o \beta$. If β is a limit, then by ind. hyp. either $\beta' \sqsubseteq_o \beta_n$ and hence $\alpha \sqsubseteq_o \lambda_n$, or β' is a limit and $\beta'_n \sqsubseteq_o \beta_n$, hence $\alpha_n \sqsubseteq_o \lambda_n$. Finally, if $\beta = \delta + 1$ is a successor, then either $\beta' \sqsubseteq_o \delta$ so that $\alpha \sqsubseteq_o \gamma + \omega^\delta \sqsubseteq_o \gamma + \omega^\delta \cdot n = \lambda_n$, otherwise by (30), β' is a successor $\delta' + 1$ with $\delta' \sqsubseteq_o \delta$, and then $(\omega^{\beta'})_n = \omega^{\delta'} \cdot n \sqsubseteq_o \omega^\delta \cdot n = (\omega^\beta)_n$, hence $\alpha_n \sqsubseteq_o \lambda_n$. □

Proposition 5 (Monotonicity)**.** *For all α, α' in Ω and n in $\mathbb{N}$,*

$$\alpha \sqsubseteq_o \alpha' \text{ implies } H^\alpha(n) \leq H^{\alpha'}(n)\,.$$

Proof. Let us proceed by induction on a proof of $\alpha \sqsubseteq_o \alpha'$, based on the deduction rules (28) and (29). For the base case, $0 \sqsubseteq_o \alpha'$ implies $H^0(n) = n \leq H^{\alpha'}(n)$ by expansiveness.

For the inductive step with (28), if $\alpha \sqsubseteq_o \alpha'$ and $\beta \sqsubseteq_o \beta'$, then

$$\begin{aligned} H^{\alpha+\beta}(n) &= H^\alpha\big(H^\beta(n)\big) && \text{(by (27))} \\ &\leq H^\alpha\big(H^{\beta'}(n)\big) && \text{(by ind. hyp. and (26))} \\ &\leq H^{\alpha'}\big(H^{\beta'}(n)\big) && \text{(by ind. hyp.)} \\ &= H^{\alpha'+\beta'}(n)\,. && \text{(by (27))} \end{aligned}$$

For the inductive step with (29), if $\alpha \sqsubseteq_o \alpha'$, then we show $H^{\omega^\alpha}(n) \leq H^{\omega^{\alpha'}}(n)$ by induction on α':

- If $\alpha' = 0$, then $\alpha = 0$ and we are done.
- If $\alpha' = \beta' + 1$ is a successor, then by (30) either $\alpha \sqsubseteq_o \beta'$, or $\alpha = \beta + 1$ with $\beta \sqsubseteq_o \beta'$. In the first case, $H^{\omega^\alpha}(n) \leq H^{\omega^{\beta'}}(n) \leq H^{\omega^{\beta'}}(H(n)) = H^{\omega^{\alpha'}}(n)$ by ind. hyp. and expansiveness. In the second case, we see by induction on $i \in \mathbb{N}$ that
$$\left(H^{\omega^\beta}\right)^i(n) \leq \left(H^{\omega^{\beta'}}\right)^i(n) \tag{33}$$
for all i and n thanks to the ind. hyp. Thus $H^{\omega^{\beta+1}}(n) = \left(H^{\omega^\beta}\right)^n(n) \leq \left(H^{\omega^{\beta'}}\right)^n(n) = H^{\omega^{\beta'+1}}(n)$ for all n, and we are done.
- If $\alpha' = \lambda'$ is a limit, then by (32) either $\alpha \sqsubseteq_o \lambda'_n$ or α is a limit λ and $\lambda_n \sqsubseteq_o \lambda'_n$. In the first case $H^{\omega^\alpha}(n) \leq H^{\omega^{\lambda_n}}(n)$ by ind. hyp.; in the second case $H^{\omega^\lambda}(n) = H^{\omega^{\lambda_n}}(n) \leq H^{\omega^{\lambda'_n}}(n) = H^{\omega^{\lambda'}}(n)$ using the ind. hyp. □

C.3 Robustness

Proposition 6 (Robustness)**.** *Let $a \geq 0$ and $x \sqsubseteq_p x'$ be two strings in P_a. Then, $H^{\eta(x)}(n) \leq H^{\eta(x')}(n')$ for all $n \leq n'$ in $\mathbb{N}$.*

Proof. We prove that $\eta(x) \sqsubseteq_o \eta(x')$ by induction on x and conclude using Prop. 5 and Eq. (26). If $x = \varepsilon$, $\eta(x) = 0 \sqsubseteq_o \eta(x')$. Otherwise we can decompose x as yza according to (14) with $y \in P_a$ and $z \in P_{a-1}$. By (3), $x' = y'z'a$ with $y \sqsubseteq_{\mathrm{p}} y'$ and $za \sqsubseteq_{\mathrm{p}} z'a$. Observe that y' and z' are in P_a, and writing $z'a = z'_1 a \cdots z'_m a$ for the canonical decomposition of z'—where necessarily each z'_j is in P_{a-1}—, then $z \sqsubseteq_{\mathrm{p}} z'_1$ as there is no other way of disposing of the other occurrences of a in z'.

By ind. hyp., $\eta(y) \sqsubseteq_o \eta(y')$ and $\eta(z) \sqsubseteq_o \eta(z'_1)$. Then, because $\eta(x) = \eta(y) + \omega^{\eta(z)}$ and $\eta(x') = \eta(y') + \omega^{\eta(z'_1)} + \cdots + \omega^{\eta(z'_m)}$, we see by (28) and (29) that $\eta(x) \sqsubseteq_o \eta(x')$. □

D Generalized Priority Embeddings

Let $d \in \mathbb{N}$ be a *priority level* and let $\gamma = (\Gamma_i, \leq_i)_{(0 \leq i \leq n)}$ be a family of wqos for some $n \geq d$, a *generalized stratified level-d priority alphabet over γ* (*generalized priority alphabet* for brevity) is $\Sigma_{d,\gamma} \stackrel{\text{def}}{=} \{(a, w) : 0 \leq a \leq d, w \in \Gamma_a\}$. Informally speaking, such an alphabet consists of alphabet symbols from the Γ_a such that each $w \in \Gamma_a$ is paired with the priority level a. A particular case is the *uniform* one, where there exists a wqo $(\Gamma, \leq)$ such that $(\Gamma_i, \leq_i) = (\Gamma, \leq)$ for all $0 \leq i \leq n$.

Example 1. Letting Γ be a finite set of messages represented as strings and $\leq$ the identity relation yields a uniform generalized priority alphabet where a priority can be assigned to each message. Such an alphabet underlies the wqo used for showing that planar planted trees are well-quasi-ordered under minors, c.f. Sec. D.5 below. Another example is $\Gamma = \Sigma^*$ for some finite alphabet Σ and where $\leq$ is the substring embedding, which allows for representing unbounded messages on a lossy channel which are tagged with a priority level.

As in the main part, we define the *generalized priority embedding* in two equivalent ways, via a string rewriting system and via factorisations.

D.1 Superseding Viewpoint

We define the *generalized priority relation* over finite strings in $\Sigma^*_{d,\gamma}$ as the transitive reflexive closure $\stackrel{*}{\to}_{\#,\gamma}$ of the string rewriting system with the following two rules schemata

$$(a, w)(a', w') \to_{\#,\gamma} (a', w') \qquad \text{if } a \leq a' \,, \tag{34}$$

$$(a, w) \to_{\#,\gamma} (a, w') \qquad \text{if } w' \leq_a w \,. \tag{35}$$

The induced ordering $\leq_{\#,\gamma}$ is now defined as $x \leq_{\#,\gamma} y \stackrel{\text{def}}{\Leftrightarrow} y \stackrel{*}{\to}_{\#,\gamma} x$.

D.2 Embedding Viewpoint

Given $x, y \in \Sigma^*_{d,\gamma}$, we define the *generalized priority embedding* $x \sqsubseteq_{p,\gamma} y$ as $x \sqsubseteq_{\mathrm{p},\gamma} y \stackrel{\text{def}}{\Leftrightarrow} x = (a_1, v_1) \cdots (a_\ell, v_\ell)$ and y can be factored as $y = y_1(a_1, w_1)y_2(a_2, w_2) \ldots y_\ell(a_\ell, w_\ell)$

with $y_i \in \Sigma^*_{a_i,\gamma}$ and $v_i \leq_{a_i} w_i$ for all $1 \leq i \leq \ell$. This embedding relation is a quasi-ordering, which can be proved in the same way as in Lem. 1.

The definition immediately yields the following properties:

$$\varepsilon \sqsubseteq_{\mathrm{p},\gamma} y \quad \text{iff} \quad y = \varepsilon, \tag{36}$$

$$x_1 \sqsubseteq_{\mathrm{p},\gamma} y_1 \text{ and } x_2 \sqsubseteq_{\mathrm{p},\gamma} y_2 \quad \text{imply} \quad x_1 x_2 \sqsubseteq_{\mathrm{p},\gamma} y_1 y_2, \tag{37}$$

$$x_1 x_2 \sqsubseteq_{\mathrm{p},\gamma} y \quad \text{implies} \quad \exists y_1 \sqsupseteq_{\mathrm{p},\gamma} x_1 : \exists y_2 \sqsupseteq_{\mathrm{p},\gamma} x_2 : y = y_1 y_2, \tag{38}$$

$$v \leq_a w \quad \text{implies} \quad (a, v) \sqsubseteq_{\mathrm{p},\gamma} z(a, w) \text{ for all } z \in \Sigma^*_{a,\gamma}. \tag{39}$$

We first show that $(\Sigma^*_{d,\gamma}, \sqsubseteq_{\mathrm{p},\gamma})$ is a quasi-ordering.

Lemma 7. *Let $\Sigma_{d,\gamma}$ be a generalized priority alphabet. Then $(\Sigma^*_{d,\gamma}, \sqsubseteq_{p,\gamma})$ is a qo.*

Proof. Reflexivity follows obviously. Regarding transitivity, let $x, y, z \in \Sigma_{d,\gamma}$ be such that $x \sqsubseteq_{\mathrm{p},\gamma} y \sqsubseteq_{\mathrm{p},\gamma} z$ and write $x = (a_1, u_1) \cdots (a_\ell, u_\ell)$. Since $x \sqsubseteq_{\mathrm{p},\gamma} y$, we can write $y = y_1(a_1, v_1) \cdots y_\ell(a_\ell, v_\ell)$, where $u_i \leq_{a_i} v_i$ and each $y_i = (b_{1,i}, v_{1,i}) \cdots (b_{m_i,i}, v_{m_i,i}) \in \Sigma^*_{a_i,\gamma}$ for all $1 \leq i \leq \ell$. Consequently, since $y \sqsubseteq_{\mathrm{p},\gamma} z$, we can decompose z as $z = z_1(a_1, w_1) \cdots z_\ell(a_\ell, w_\ell)$, where each z_i is of the form

$$z_i = z_{1,i}(b_{1,i}, w_{1,i}) \cdots z_{m_i,i}(b_{m_i,i}, w_{m_i,i}) z'_i.$$

Since each $(b_{j,i}, w_{j,i}) \in \Sigma_{a_i,\gamma}$, by definition of $\sqsubseteq_{\mathrm{p},\gamma}$ we have $z_i \in \Sigma^*_{a_i,\gamma}$, hence the above decomposition of z yields $x \sqsubseteq_{\mathrm{p},\gamma} z$. □

Moreover, $\leq_{\#,\gamma}$ and $\sqsubseteq_{\mathrm{p},\gamma}$ coincide, as shown by the next lemma.

Lemma 8. *For any $x, y \in \Sigma^*_{d,\gamma}$, $x \leq_{\#,\gamma} y$ iff $x \sqsubseteq_{p,\gamma} y$.*

Proof. In the following, write x as $x = (a_1, v_1) \cdots (a_k, v_k)$. Suppose $x \leq_{\#,\gamma} y$, i.e. $y \xrightarrow{*}_{\#,\gamma} x$. We show the statement by induction on the number of rewrite steps. For the induction step, let $y \to_{\#,\gamma} z$ such that $z \xrightarrow{*}_{\#,\gamma} x$. By the induction hypothesis, $x \sqsubseteq_{\mathrm{p},\gamma} z$, i.e., z can be factored as $z = z_1(a_1, w_1) \cdots z_k(a_k, w_k)$ such that $z_i \in \Sigma^*_{a_i,\gamma}$ and $v_i \leq_{a_i} w_i$ for all $1 \leq i \leq k$. We do a case distinction on which rewriting rule is applied. If $y \to_{\#,\gamma} z$ via (34) then y is obtained from z by replacing some $z_j = z_{j,1} \cdots z_{j,\ell_j}$ with $z'_j = z_{j,1} \cdots z_{j,i-1}(b, w) z_{j,i} \cdots z_{j,\ell_j}$ for some $1 \leq i \leq \ell_j$ and (b, w) such that in particular $b \leq a_j$. Thus, y factors as $y = z_1(a_1, w_1) \cdots z'_j(a_j, w_j) \cdots z_k(a_k, w_k)$, which by definition gives $x \sqsubseteq_{\mathrm{p},\gamma} y$. Otherwise, if $y \to_{\#,\gamma} z$ via (35), y is obtained by replacing some (a, w) occurring in z with (a, w') for some $w' \geq_a w$. By transitivity of $\leq_a$, $x \sqsubseteq_{\mathrm{p},\gamma} y$ follows immediately.

Conversely, assume $x \sqsubseteq_{\mathrm{p},\gamma} y$ and thus y factors $y = y_1(a_1, w_1) \cdots y_k(a_k, w_k)$. Since for every (a, w) occurring in some y_i we have $a \leq a_i$, by repeatedly applying (34) we have $y \to^*_{\#,\gamma} z = (a_1, w_1) \cdots (a_k, w_k)$. Moreover, $v_i \leq_{a_i} w_i$ for all $1 \leq i \leq k$, and thus by repeated application of (35) we get $z \to^*_{\#,\gamma} x$, as required. □

D.3 Generalized Priority Embedding is a WQO

Our main result of interest is that generalized priority embeddings establish a wqo.

Theorem 25. *Let $\Sigma_{d,\gamma}$ be a generalized priority alphabet. Then $(\Sigma^*_{d,\gamma}, \sqsubseteq_{p,\gamma})$ is a wqo.*

Proof. We proceed by induction on d. For the induction step, any $x \in \Sigma^*_{d,\gamma}$ can be uniquely factored as

$$x = x_0(d, v_1) \cdots x_{m-1}(d, v_m)x_m$$

such that $x_i \in \Sigma^*_{d-1,\gamma}$ for all $0 \leq i \leq m$. By the induction hypothesis, $(\Sigma^*_{d-1,\gamma}, \sqsubseteq_{\mathrm{p},\gamma})$ is a wqo. We define an *order reflection* $r\colon \Sigma^*_{d,\gamma} \to \Theta_{d,\gamma}$, where

$$\Theta_{d,\gamma} \stackrel{\text{def}}{=} \Sigma^*_{d-1,\gamma} + \Sigma^*_{d-1,\gamma} \times (((\{d\} \times \Gamma_d) \times \Sigma_{d-1,\gamma})^* \times (\{d\} \times \Gamma_d) \times \Sigma^*_{d-1,\gamma}.$$

Since $\Theta_{d,\gamma}$ is obtained from the well-quasi orders $(\Sigma^*_{d-1,\gamma}, \sqsubseteq_{\mathrm{p},\gamma})$, $(\Gamma_d, \leq_d)$ and equality on $\{d\}$ by disjoint sum, Cartesian product and substring embedding, this allows us to conclude that $(\Sigma^*_{d,\gamma}, \sqsubseteq_{\mathrm{p},\gamma})$ is a wqo. To this end, for x and m as above such that $m > 0$, define

$$r(x) \stackrel{\text{def}}{=} (x_0, (((d, v_1), x_1) \cdots ((d, v_{m-1}), x_{m-1})), (d, v_m), x_m), \tag{40}$$

and $r(x) \stackrel{\text{def}}{=} x_0$ if $m = 0$. We need to verify that whenever $r(x) \leq r(y)$ wrt. the ordering $\leq$ associated with $\Theta_{d,\gamma}$ then $x \sqsubseteq_{\mathrm{p},\gamma} y$. This is obvious when both $r(x) = x$ and $r(y) = y$. Otherwise, let $r(x)$ be as in (40) and write

$$r(y) = (y_0, (((d, w_1), y_1) \cdots ((d, w_{n-1}), y_{n-1})), (d, w_n), y_n).$$

Since $r(x) \leq r(y)$, through the subword ordering there exist indices $i_1, \ldots, i_{m-1}$ such that for

$$u = (y_0, (((d, w_{i_1}), y_{i_1}) \cdots ((d, w_{i_{m-1}}), y_{i_{m-1}})), (d, w_n), y_n),$$

we have $r(x) \leq u$. By assumption, $x_0 \sqsubseteq_{\mathrm{p},\gamma} y_0$, $x_m \sqsubseteq_{\mathrm{p},\gamma} y_n$, and furthermore $v_j \leq_d w_{i_j}$ and $x_j \sqsubseteq_{\mathrm{p},\gamma} y_{i_j}$ for all $1 \leq j < m$. By repeatedly applying (37) and (39), we get

$$(d, v_j)x_j \sqsubseteq_{\mathrm{p},\gamma} (d, w_{i_{j-1}+1})y_{i_{j-1}+1} \cdots \sqsubseteq_{\mathrm{p},\gamma} (d, w_{i_j})y_{i_j}$$

for all $1 \leq j \leq m$, where $i_0 \stackrel{\text{def}}{=} 0$ and $i_m \stackrel{\text{def}}{=} n$. Consequently, $x \sqsubseteq_{\mathrm{p},\gamma} y$ as required. □

D.4 Reflecting Bounded-Depth Trees

Thanks to generalized priority embeddings, we can extend the reflection of Section 4 to handle trees labeled by elements of some wqo $(\Gamma, \leq)$. This is done simply by employing a uniform generalized priority alphabet $\Sigma_{d,\Gamma}$, i.e. by setting $\gamma(i) = (\Gamma, \leq)$ for all i, and by defining the reflection $s_d\colon T_d(\Gamma) \to \Sigma_{d,\Gamma}$ through

$$s_d(f(t_1 \cdots t_n)) \stackrel{\text{def}}{=} \begin{cases} (d, f) & \text{if } n = 0, \\ s_{d-1}(t_1)(d, f) \cdots s_{d-1}(t_n)(d, f) & \text{otherwise.} \end{cases} \tag{41}$$

The corresponding notion of strong tree embeddings uses a single step

$$C[f(t_1 \cdots t_{i-1}t_{i+1} \cdots t_n)] \sqsubset^1_T C[g(t_1 \cdots t_{i-1}t_i t_{i+1} \cdots t_n)] \tag{42}$$

whenever $f \leq g$ in Γ. We leave as an exercise to the reader to check that s_d is an order reflection from $(T_d(\Gamma), \sqsubseteq_T)$ to $(\Sigma^*_{d,\gamma}, \sqsubseteq_{\mathrm{p}})$ as in Prop. 3.

D.5 Relationship to Tree Minors

Gupta gives in (Gupta, 1992) a constructive proof that finite rooted trees with an ordering on the children of every internal vertex (called *planar planted trees*) are well-quasi-ordered under minors. Recall that t_1 is a *minor* of t_2 if t_1 can be obtained from t_2 by a series of edge contractions, e.g. in the figure below repeated from Figure 2, the left tree is a minor of the right one.

Gupta provides in (Gupta, 1992) an effective linearisation which essentially associates with every tree t a word $\tau(t)$ over the uniform generalized prioritised alphabet $\Sigma_{d,\gamma}$, where d is the number of vertices of t and $\gamma = (\Gamma, =)$ with $\Gamma = \{v_1, v_2, v_3\}$.

Given $x = \tau(t_1) = (a_1, w_1)(a_2, w_2)\dots(a_k, w_k) \in \Sigma^*_{d,\gamma}$ and $y = \tau(t_2)$, he shows that t_1 is a minor of t_2 if x embeds in y (written $x \sqsubseteq_{\mathrm{g}} y$) as follows: y can be factored as $y = y_1 y_2 \cdots y_k$ such that $y_i \in \Sigma^*_{a_i,\gamma}$ and (a_i, w_i) is a substring of y_i for all $1 \leq i \leq k$. This ordering is closely related to ours. In fact, it is easily seen that $\sqsubseteq_{\mathrm{p},\gamma}$ can be viewed as a sub-structure of $\sqsubseteq_{\mathrm{g}}$, as $x \sqsubseteq_{\mathrm{p},\gamma} y$ implies $x \sqsubseteq_{\mathrm{g}} y$. Thus, our Thm. 25 yields as a byproduct that $\sqsubseteq_{\mathrm{g}}$ is a wqo.

E Maximal Order Types

The *maximal order type* of a wqo $(X, \leq)$ is a measure of its complexity defined by de Jongh and Parikh (1977) as the maximal order type of its linearizations: a *linearization* $\prec$ of $\leq$ is a total linear ordering over X that contains $\leq \setminus \geq$ as a subrelation. Any such linearization of a wqo is well-founded and thus isomorphic to an ordinal, called its order type, and the maximal order type is therefore the maximal such ordinal.

De Jongh and Parikh provide formulæ to compute the maximal order types of wqos based on their algebraic decompositions as disjoint sums, cartesian products, and Kleene star—using respectively the sum ordering, the product ordering, and the subword embedding ordering—: for wqos A and B of maximal order types in ε_0,

$$\begin{aligned} o(A + B) &= o(A) \oplus o(B) \\ o(A \times B) &= o(A) \otimes o(B) \\ o(A^*) &= \begin{cases} \omega^{\omega^{o(A)-1}} & \text{if } A \text{ is finite,} \\ \omega^{\omega^{o(A)}} & \text{otherwise.} \end{cases} \end{aligned}$$

Here, the $\oplus$ and $\otimes$ operations are the *natural sum* and *natural product* on ordinals, defined for ordinals in CNF in ε_0 by

$$\sum_{i=1}^{m} \omega^{\beta_i} \oplus \sum_{j=1}^{n} \omega^{\beta'_j} \stackrel{\text{def}}{=} \sum_{k=1}^{m+n} \omega^{\gamma_k}\,, \quad \sum_{i=1}^{m} \omega^{\beta_i} \otimes \sum_{j=1}^{n} \omega^{\beta'_j} \stackrel{\text{def}}{=} \bigoplus_{i=1}^{m} \bigoplus_{j=1}^{n} \omega^{\beta_i \oplus \beta'_j}\,, \tag{43}$$

where $\gamma_1 \geq \cdots \geq \gamma_{m+n}$ is a reordering of $\beta_1, \dots, \beta_m, \beta'_1, \dots, \beta'_n$.

An immediate consequence of the definition of a maximal order type is that, if $(A, \leq_A)$ reflects $(B, \leq_B)$, then $o(A) \geq o(B)$.

Additional References

de Jongh, D.H.J. and Parikh, R., 1977. Well-partial orderings and hierarchies. *Indagationes Mathematicae*, 39(3):195–207. doi:10.1016/1385-7258(77)90067-1.